

digit **FastTrack**

YOUR HANDY GUIDE TO EVERYDAY TECHNOLOGY

To **Developing**

Chrome Apps

Introduction to **Chrome web store**
and web apps

Web Apps using **open**
web technologies

Building an app

Hosted apps and
packaged web apps

Best apps

Uploading and
Publishing

Advantages of hosting on
Chrome Web Store



Chrome is secure.



chrome by Google

Download now at www.google.co.in/chrome



CHROME WEB STORE AND WEB APPS

powered by



CHAPTERS

CHROME APPS | JULY 2011

01 CHAPTER

Introduction to the Chrome Web Store

We take you through the idea behind the Chrome Web Store and what it takes to build a great app.

02 CHAPTER

Advantages of hosting on the Chrome Web Store

Why you should host your apps on the Chrome Web Store.

03 CHAPTER

Web apps using open web technologies

Explore the advantages of open web technologies and exploit them to your advantage while developing your dream web app.

04 CHAPTER

Hosted apps and packaged apps

A guide to choosing and implementing the right kind of app.

05 CHAPTER

Understanding Manifest

A closer look at the special features of Chrome Web Store.

CREDITS

The People Behind This Book

EDITORIAL

Editor

Robert Sovereign-Smith

Head-Copy Desk

Nash David

Writers

Ankur Mour, Meghnil Pagrut,

Piyush Waradpande

DESIGN

Sr. Creative Director

Jayan K Narayanan

Art Director

Binesh Sreedharan

Associate Art Director

Anil VK

Sr. Visualisers

PC Anoop

Senior Designers

Baiju N V, Chander Dange,

Vinod Shinde

06

CHAPTER

Uploading and publishing

Now that you've prepared the manifest, we show you how to pack your apps and get them up and running on the Chrome Web Store.

07

CHAPTER

Best apps

A collection of must-have apps for everyone.

08

CHAPTER

Your first app

Here we present the source code along with explanation for writing a basic extension without any fancy features.

09

CHAPTER

Summary

Guidelines to follow so that you can make the most of your app.

© 9.9 Mediaworx Pvt. Ltd.

Published by 9.9 Mediaworx

No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means without the prior written permission of the publisher.

July 2011

Free with Digit. Not to be sold separately. If you have paid separately for this book, please email the editor at editor@thinkdigit.com along with details of location of purchase, for appropriate action.



COVER DESIGN: MANJITH PB

INTRODUCTION

The Chrome Web Store is a new online marketplace, where Google Chrome users can discover thousands of web apps for their browser.

In this Fast Track, we'll first explain what the Chrome Web Store is and what exactly web apps are. Then, we'll explore the benefits of listing your web apps in the Chrome Web Store. After this, we'll review the open web technologies that Chrome supports and discuss how you can use these capabilities to develop great web apps.

Next, we'll go over the differences between the types of web apps you can list in the Chrome Web Store. We'll also show you how to upload and publish your app with an easy to follow step-by-step tutorial.

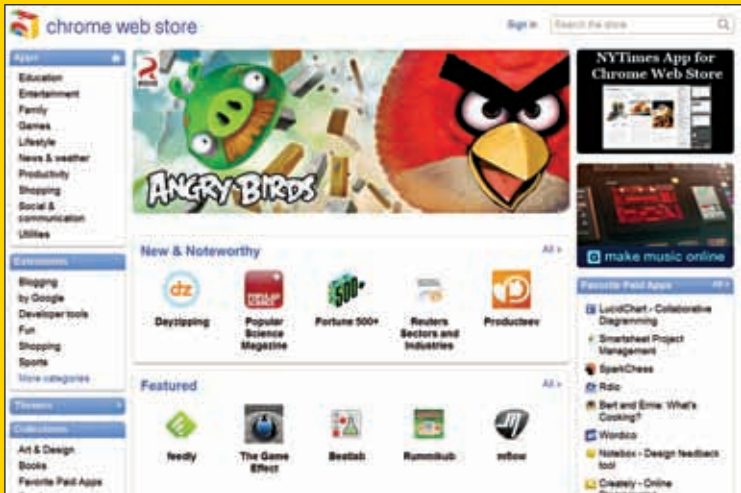
In addition, to inspire you, we've compiled a collection of the best Chrome Web Store apps. Finally, we'll also walk you through the architecture of a simple demo app and show you how it was built.

Enjoy!

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



CHAPTER #1



INTRODUCTION TO THE CHROME WEB STORE AND WEB APPS

We take you through the idea behind the Chrome Web Store and what it takes to build a great app.

Since its announcement at the Google I/O conference on May 19, 2010, the Chrome Web Store has attracted developers and users alike. Released on December 6, 2010, the Chrome Web Store is an online marketplace that gives you access to thousands of web apps that can help the more than 160 million chrome users customise their browsing experience. All you need to do is install web apps from the Chrome Web Store.

What are web apps?

Over the years, web sites have evolved from pages of static content to interactive applications that are similar to software you'd normally install from a CD. A good example of a web app is Gmail. Gmail provides you with the functionality you'd normally expect to get from a desktop app but it comes with all the benefits of web apps. For example, you don't need to manually update Gmail—you always have the most up-to-date version. And if you use multiple devices to email—for example, a work and a personal computer or even a phone—you don't have to reinstall it every time; you just need to fire up your browser.

Most web browsers make no distinction between apps and web pages. Google Chrome is different in this regard as it supports the concept of installable web apps. An installable web app can be a normal website with a bit of extra metadata; this type of app is called a hosted app. Alternatively, an installable web app can bundle all its content into an archive that users download when they install the app; this is a packaged app.



Web Store:
Enhance your
Chrome experience
with Apps from the
Chrome Web Store

Both hosted and packaged apps have shortcut icons in Chrome's New Tab page, making web apps easily accessible to Chrome users who install them.

Creating a hosted or packaged app for your web service doesn't come with just a placement of your app icon in Chrome's New Tab page. You also benefit by an improved security settings system that was designed with web apps in mind. While normal web sites need to seek your permission every time they perform increasingly common functions (such as showing notifications, using the clipboard or using your location to offer a more customised experience) installable web apps need to ask for these permissions only once—when you install them to Google Chrome. This makes for a better user experience.

And the best part? Because Chrome is updated regularly, it supports all the latest bells and whistles from the open web platform, including HTML5 APIs and WebGL. This means that you don't have to wait to use these technologies; you can integrate them in your web services and deploy them without compatibility scares to the 160 million people who are all using the same version of Chrome.

Getting started

Think the app way

What could your app do?

Your app could make it easier for the user to send text messages, or help students get updates on their results. Either way, the success of your app depends on how useful it is to the common user. Using open web technologies such as notifications, geolocation, and WebGL for stunning 3D graphics, you can make your app stand out in the crowd.

How should your app do it?

After you have your product vision well defined, you can think about the form factor of your app. You can either choose to package it as a downloadable file or run it off the web; either has its own advantages and limitations. We will look into each of this as we go along.

Before you actually start building your app, here are a couple of things you should keep in mind.

► Remember what your app does

If your app bombards the user with different features, he/she might find it complicated or annoying to use. So, work towards exclusivity, creativity and innovation. In fact, it is popularly believed that judgements are made

within 30 seconds of using an app.

The purpose of your app doesn't really matter as long as it delivers what the user wants, in the best and the fastest way possible. Don't you just hate it when you find a music streaming app that makes you create a new account or do a random survey, even before you get to enjoy the first song? Make sure your app connects users directly to their needs. Some of the best services in the world have been successful only because they put user satisfaction above everything else.

Although you are encouraged to have features, make sure they don't overshadow the core function. While useful features will make your app unique, unrelated features will depreciate its value in the long run.

As an app developer, you should rate user experience very highly on your plan. Using open web technologies that are supported by modern browsers, you can enhance user experience by offering services such as instant notifications, geolocation, beautiful web fonts, drag-and-drop and high quality graphics that will delight your users. They can use cross-origin resource sharing to consume content from remote sites, and WebSockets to show live content and support real-time communication across the web. By implementing the above features and innovations, your app will not only meet users' expectations, but exceed them!

► **UI and design**

We've all seen how popular app-driven environments have multiple apps around the same function. If you have 10 different apps to choose from, just to manage your Twitter updates, how do you select an app? Popular choice is quite significantly driven by superior UI and design quality.

The web apps that you design will run on a desktop or a laptop. Unlike a mobile device, you have a larger screen real estate to spare for your UI on a PC or laptop, which is further enhanced by the full screen feature in Chrome. With all the extra space, you might even do the opposite of a small-screen mobile app: instead of shrinking the normal browser view with cluttered information, present your user with information that is spaced out and pleasant to access. This way, you'd ensure an enriching user experience for your app.

Now, since most users would install your app after weighing its advantages, don't turn them off with unnecessary promotions. In fact, this could discourage them from recommending your app to others. As an example, once inside a music app, a user should not see banners explaining the benefits of your app. The user has already chosen and installed the app,

and as such does not need a sales pitch again. You could, however, inform them about the latest updates available for your app.

A user expects to be within the web app, with all UI elements dedicated to the task at hand. The app exists in its own context and navigation scheme, and is external and independent from the web site's context or navigation scheme. When using a specific web app, a user is only aware of that particular app. When designing the web app, remember that it is an independent application, and not a part of a web site. Keep all relevant UI elements on top, and get rid of all unrelated ones.

A web app is completely different from the usual pattern of a logo in the header, menu on the side, detail links in the footer, and so on. Clear your mind of all web site designs and stress on user requirements; even better, think about how you'd want an app like that to look and feel like.


While making the UI as simple as possible, it often ends up dull. A good web app puts a premium on aesthetics without sacrificing usability.

Most modern browsers support Scalable Vector Graphics (SVG), canvas (a 2D drawing surface), CSS3 transforms and animations, new colour models, integrated native video, and professional fonts. There are even proposals for integrating 3D engines into browsers. Because these building blocks are all native to the browser, it is very easy to combine them for greater effect. For example, you can add rounded corners to video or make a CSS animation that uses new colour schemes.

With the UI and design taken care of, you can look at the following factors while developing your app.

► **Speed**

The importance of how fast your app delivers results can never be over-emphasised. None of us like to wait for a response (do we?). The responsiveness of a web app, both real and perceived, should be as fast as any normal application.

A web app is expected to be fast to match up to the speed at which the user works. Each second that he/she has to wait, translates to money you stand to lose. Research has shown that faster web apps make more money and rank higher in search results. A good web app feels and acts fast. 

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



ADVANTAGES OF HOSTING ON THE CHROME WEB STORE

A look at the advantages of hosting on the Chrome Web Store and why it stands out from the others.

How to get started

To get started with apps for the Chrome Web Store you'll need to make the following key decisions:

What should your app do?

What do you actually want to provide for users? This is something that only you can answer, but you might want to think about how you can service your current or potential users better by using all the latest open web technologies. Things like notifications, geolocation and WebGL for beautiful 3D graphics rank high in the priority list of lots of developers and can make your app stand out.



Discovery: Chrome Web Store's Search lets you find all the apps you need

How should your app do it?

After you have your product vision well defined, you need to think about the form factor of your app. You can choose either a packaged app or a hosted app; either format has its own advantages and limitations. We will look into each format in the following chapters.

How will you monetize your app?

There isn't going to be a one-size-fits-all answer in this area. You need to make a choice based on your current business model. The Chrome Web Store supports free and paid apps:

Free:

- ▶ **Free, ad-supported:** You can distribute your apps at the store for free but show ads in them. There are plenty of ad networks you can use to do so, or you can sell the space in your app directly as a sponsorship.
- ▶ **Free, referral-based:** You can choose to use referral-based advertising directing users to e-commerce sites, for example.

Paid:

If you operate only in India, you will need to use a custom payment solu-

tion that works in India, such as CCAvenue or payment gateways by popular banking and financial institutions such as HDFC Bank, ICICI Bank and Citibank.

If you have a bank account in countries where Google Checkout is supported you can use the store's built-in payment systems:

- ▶ **One-time purchase:** You can sell your product and service to users with a one-time fee.
- ▶ **Subscription:** You have the option to sell a subscription to your product or service.
- ▶ **Freemium:** You can deliver a minimum level of functionality that allows users to try your product or service and then decide to upgrade to a more full-featured version for a fee.

Best Practices

As you start building your app, here are a couple of things you should keep in mind:

Stay focused

Don't start adding hundreds of features. This can confuse your users, especially if they are new. Instead, try to have your app do one thing very well. If you need to create more features, make sure they do not overshadow the main functionality that attracted users to your app in the first place.

Think about the login experience

When a Chrome user decides to install an app, he or she has already decided they want to try out your service. So, do not lead them to a marketing or a registration page. After all, don't you hate it when you want to try a new music app but need to spend 10 minutes to create an account before you can listen to a single song?

Instead, try to get users inside your app immediately: let them use it to understand its value. If you need users to register you can always ask them to complete this task after giving them a chance to experience your app. It is also highly recommended to provide support for Google accounts. Most of your users will already be logged into a Google account whenever they use your app. Another good idea is to correlate the Google account to the user account in your system by storing the user's OpenID credentials from Google's OpenID service.

Make technology “transparent”

Your app should just work. You can enhance it with instant notifications, geolocation, beautiful web fonts, drag-and-drop and high quality graphics but also these should be integrated in the experience you offer, rather than being exposed separately to users. For example, Weather Underground gives users information about the weather conditions in their area using geolocation, without asking users to take any action to enable this feature.

Invest in design

If you have 10 apps to choose from to, let's say, manage your Twitter updates, how do you select one to use? Most users often make their decisions based on the design quality of an app. With apps in Chrome you can create a rich user experience:

- ▶ **Big screen:** Unlike apps for a mobile device, apps in Chrome have access to larger screen real estate. With all the extra space, instead of shrinking the normal browser view with cluttered information, you can present your user with information that is spaced out and pleasant to access.
- ▶ **New tools:** You now have a rich array of options for designing and authoring highly graphical and beautiful applications. Most modern browsers support SVG, canvas, CSS3 transforms and animations, new colour models, integrated native video, and professional fonts. Because these building blocks are all native to the browser, it is very easy to combine them for greater effect. For example, you can add rounded corners to video or make a CSS animation that uses new colour schemes.
- ▶ **Innovation:** A web app is completely different from the usual pattern of a logo in the header, menu on the side, detail links in the footer, and so on. Clear your mind of all website designs and stress on user requirements; even better, think about how you'd want an app like that to look and feel like.

Don't forget usability

A good web app puts a premium on aesthetics without sacrificing usability. Specifically, a user expects to be within the web app, with all UI elements dedicated to the task at hand. The app exists in its own context and navigation scheme, and is external and independent from another website's context or navigation scheme.



Note taking just got simpler

When using a specific web app, a user is only aware of that particular app. When designing the web app, remember that it is an independent application, and not a part of a website. Keep all relevant UI elements on top, and get rid of all unrelated ones.

Finally, your interface should have minimal levels of menu options to reach its key functionality.

Remember that speed matters

The importance of how fast your app delivers results can never be over-emphasised. None of us like to wait for a response. The responsiveness of a web app, both real and perceived, should be as good as any desktop application. A web app is expected to be fast to match the speed at which the user works. Each second that the user has to wait translates to money that you stand to lose. Research has shown that faster web apps make more money and rank higher in search results. A good web app feels and acts fast.

Optimize your store listing

Be creative and descriptive when it comes to choosing your app's name, writing the description and designing your logo. Include plenty of screenshots and make sure you include in your listing the large and small promotional icons that will allow the Chrome team to promote your app.

Finally, be very careful in choosing your app's category. The Chrome Developer Dashboard will let you specify up to two categories for your

app. Keep in mind that the categories aren't final, and watch out for new categories that might be added and could be a better fit for your app.

Additional reading

Design assets for your app:

<http://code.google.com/chrome/webstore/docs/images.html>

Branding guidelines:

<http://code.google.com/chrome/webstore/branding.html>

Developer agreement:

<http://code.google.com/chrome/webstore/terms.html>

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



WEB APPS USING NEW OPEN WEB TECHNOLOGIES

Explore the advantages of open web technologies and exploit them to your advantage while developing your dream web app.

HTML5 forms

Not many people get excited about forms, but HTML5 brings some big improvements, both for the developers creating them and for the users filling them out. New form elements, attributes, input types, browser-based validation, CSS3 styling techniques, and the FormData object make it easier and hopefully more enjoyable to create forms.

Frequently, mobile devices will customize their on screen keyboards to provide a more streamlined user experience. One of the best “features” of these new form elements, is that they have a reasonable fallback experience in browsers that don’t support them. Instead of showing the specialized input fields, old browsers simply allow the user to provide the data in a normal text box.

NEW INPUT TYPES

HTML5 introduces 13 new input types. When viewed in a browser that doesn't support them, these input types fall back to text input.

Input Type	Purpose	Notes
tel	For entering a telephone number.	tel does not enforce a particular syntax, so if you want to ensure a particular format, you can use <code>pattern</code> or <code>setCustomValidity()</code> to do additional validation.
search	To prompt users to enter text that they want to search for.	The difference between search and text is primarily stylistic. Using an input type of search might result in the input field being styled in a way that is consistent with that platform's search fields.
url	For entering a single URL.	url is intended for entering a single, absolute URL, which represents a pretty wide range of values.
email	For entering either a single email address or a list of email addresses.	If the multiple attribute is specified, then multiple email addresses can be entered, separated by commas.
datetime	For entering a date and time with the time zone set to UTC.	
date	For entering a date with no time zone.	
month	For entering a date with a year and a month, but no time zone.	
week	For entering a date that consists of a week-year number and a week number, but no time zone.	An example of this format is 2011-W05 for the fifth week of 2011.
time	For entering a time value with hour, minute, seconds, and fractional seconds, but no time zone.	
datetime-local	For entering a date and time with no time zone.	
number	For numerical input.	Valid values are floating point numbers.
range	For numerical input, but unlike number, the actual is not important.	The implementation of the range control is a slider in most browsers that support it.
color	For choosing color through a color well control.	The value must be a valid lower-case simple color such as #ffffff.

CSS

With the introduction of CSS3 it has never been easier to create rich and beautiful sites and applications in HTML. There are many new technologies and extensions to CSS3 including: 2D Transformations, Transitions, 3D Transforms and WebFonts to name just a few.

You can create rich user experiences with no coding effort required, just simply apply a little CSS to your existing applications. CSS3 Transformations allow you to apply rotations, scales, skews and translations to any DOM element with simple styles. Transitions allow you to specify how the style on a DOM element will animate between transformed states. For example, you can animate the background color when the user hovers over an element. Applying 3D transforms is simple too. Just specify the transformation in 3d co-ordinates and you are ready.

Things that developers and designers have been asking for for years are also included, like rounded corner support with border-radius, opacity, new colour models like HSLA and RGBA, text shadows, box shadows and plenty more. There are lots of great examples of people pushing the edge with nothing but CSS and HTML!

Canvas

The HTML5 canvas element and API is a way to use scripting to draw graphics on a web page, with control down to the pixel level. The canvas element has several methods for drawing paths, boxes, circles, characters, and adding images.

It can be used to draw graphs, edit or make photo compositions or do simple and complex animations. With knowledge of HTML and JavaScript, you've got enough to get started.

```

• <canvas id="canvas1" width="400" height="400"></canvas>
• function draw()
• {
•   var canvas = document.getElementById("canvas1");
•   var ctx = canvas.getContext("2d");
•   ctx.fillStyle = "rgb(100,0,0)";
•   ctx.fillRect (10,10,55,50);
•   ctx.fillStyle = "rgba(0,50,200,0.2)";
•   ctx.fillRect (30,30,55,50);
• }

```

To draw on a canvas element, we need to use the `getContext("2d")` method to get the drawing surface. The 2D context represents a flat Cartesian surface, where the (0,0) origin is at the top left corner, X and Y increasing as to move right or down respectively. The function `draw()` assigns the 2D context to the variable `ctx`.

The example then draws a red rectangle, and a blue-green rectangle by setting the fill color via `ctx.fillStyle` and then drawing the rectangle with `fillRect`.

This was a pretty basic example, and there are a lot more possibilities than drawing two rectangles. To see about what Canvas can really do, check out the Canvas tutorials on HTML5Rocks.com at <http://www.html5rocks.com/tutorials/#canvas>

Audio and Video: The new way!

In the past, rich media to your site meant depending on a plug-in, but HTML5 promotes audio and video to first class citizens, allowing them to be included on your page in the same way you'd include an ``. While this section focuses on video, the concepts for `<audio>` are exactly the same, just without moving pictures!

Video Element

The video element is one of those HTML5 features that gets a lot of attention. Although it's recently joined the rest of the ubiquitous HTML tags, its capabilities and support across browsers have increased at an amazing speed. One of it's biggest advantages is the natural integration with the other layers of the web development stack such as CSS and JavaScript as well as the other HTML tags.

- `<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="425" height="344" codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,40,0">`
- `<param name="allowFullScreen" value="true" />`
- `<param name="allowscriptaccess" value="always" />`
- `<param name="src" value="http://www.youtube.com/v/oHg5SJYRHA0&hl=en&fs=1" />`
- `<param name="allowfullscreen" value="true" />`
- `<embed type="application/x-shockwave-flash"`


```
width="425" height="344" src="http://www.youtube.com/v/
oHg5SJYRHA0&hl=en&fs=1&" allowscriptaccess="always"
allowfullscreen="true">
• </embed>
• </object>
```

But with HTML5 you can do this with minimal coding,

```
• <video width="640" height="360" src="http://www.youtube.
com/demo/google_main.mp4" controlsautobuffer>
• <p> Try this page in Safari 4! Or you can <a
href="http://www.youtube.com/demo/google_main.mp4">download
the video</a> instead.</p>
• </video>
```

Now let us dig deeper and garner a basic understanding of the video tag.

The Markup

We can include an HTML5 video element simply by using the following:

```
• <video width="640" height="360" controls>
• <source src="http://www.youtube.com/demo/google_main.mp4"
/>
• <p>Sorry, your browser doesn't support video.</p>
• </video>
```

You can include multiple `<source>` tags in your video element to provide support for multiple video codecs. Don't forget to ensure the video file is served with the right MIME type.

Video formats

Think of a video format as a ZIP file which contains the encoded video stream and audio stream. The three formats you should care about for the web are:

- .webm = VP8 + Vorbis
- .ogg/.ogv = Theora + Vorbis
- .mp4 = H.264 + AAC

By providing support for multiple video formats, you can be sure that your video will run in any browser, on any platform.

```
• <video>
• <source src="movie.webm" type='video/webm; codecs="vp8,
```

```
vorbis"" />
```

- `<source src="movie.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />`
- `<source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"' />`
- Video tag not supported. Download the video `here`.
- `<video>`

Encoding

There are many free and paid video encoders available that will help you get your video into the appropriate formats. Check out <http://diveintohtml5.org/video.html#webm-cli> for some good tools you can use.

Attributes (HTML)

▶ AUTOPLAY

The video tag has an attribute that allows the video to play when the page loads.

- `<video src="abc.mov" autoplay>`
- `</video>`

▶ DOWNLOAD

If the browser doesn't know what to do with video tag, or if there is a display error, you can offer a download to the video instead:

▶ AUTOBUFFER

Autobuffer attribute is used when autoplay is not used but the page/site owner thinks the video will be watched at some point. The video is downloaded in the background, so when the user starts the video, it will be able to play at least some of the content. If both autoplay and autobuffer are used, then autobuffer is ignored.

▶ POSTER

You can use the poster attribute to display a frame of the video (as a .jpg, .png, whatever), in case the video doesn't load for some reason.

▶ CONTROLS

Adding this attribute means you can use your own play/pause/etc buttons for your video. It is used to indicate that instead of building our custom controls we want the browser to render one automatically for us.

Integration

As we said in the introduction, the main advantage of the video tag being a member of the HTML5 family, is the integration with the other layers of web development stack. Let us take a look at the kind of interesting stuff you can do

▶ VIDEO + other HTML

All the common HTML attributes, we described in the previous section, can be used in the video player.

```

• <video poster="star.png" autoplay loop controls tab-
  index="0">
•   <source src="movie.webm" type='video/webm; codecs="vp8,
  vorbis"' />
•   <source src="movie.ogv" type='video/ogg; codecs="theora,
  vorbis"' />
• </video>

```

The full level of integration of the video tag as a native browser element deprecates issues that might arise with third party embedded players such as drop down menus and iframes overlaying on the player or weird layout behaviour when the other HTML elements surrounding the video were dynamically resized.

Since the video is not treated as an embedded foreign object, there are some other benefits that affect the user experience. For instance, even if the focus is on the player itself actions like page scrolling or keyboard key strokes will be completely functional.

▶ VIDEO + JS

The video tag comes with a set of attributes and methods which let you have a fine control of your video from your JS code. As any other HTML element you can attach common events to the video tag such as drag events, mouse events, and focus events. Since loading a video resource can have many caveats, there are many events to have a fine control of the loading process, both at network level (loadstart, progress, suspend, abort, error, emptied, stalled) and at buffering level (loadedmetadata, loadeddata, waiting, playing, canplay, canplaythrough).

At its simplest level, you can attach the canplay event to start doing stuff with the video.

```

• video.addEventListener('canplay', function(e) {
•   this.volume = 0.4;

```

- `this.currentTime = 10;`
- `this.play();`
- `}, false);`

• VIDEO + CSS

The video tag can also be styled using traditional CSS (e.g. border, opacity, etc) since it is a first-class citizen in the DOM. But the cool thing is that you can even style it with the latest CSS3 properties like reflections, masks, gradients, transforms, transitions and animations.

▶ VIDEO + CANVAS

Canvas holds loads of possibilities when used in conjunction with the video tag. You can make use of the following features :

- ▶ Import images - The `drawImage` method lets you import images from three different sources: an image element, another canvas element and a `<video>` element!
- ▶ Export Images - The `toDataURL` method lets you export the content of the canvas to an image
- ▶ You can even hide the canvas you use to import/export.
- ▶ You can increase the frequency at which you import and render the image from the video in order to emulate the same video frame rate in the canvas. This lets you do all sorts of fancy transforms in the canvas as if you were doing it in the video.
- ▶ You can use the same technique of importing images to WebGL where you will be able, for instance, to import the frames of a video and render them on a spinning 3D cube.

▶ VIDEO + SVG

SVG images and their behaviour are defined in XML text files. SVG provides a programmatic way to render and manipulate vector graphics and comes with features such as SVG filter effects. With these filters you can target a specific DOM element and apply some out of the box effects such as blur, composite and tiles.

One advantage of the video tag, being a member of the HTML5 family, is the integration with the other layers of web development stack. To learn more about how to integrate video into your site, and other parts of the web stack, be sure to check out the HTML5 video tutorial on [HTML5Rocks.com](http://www.html5rocks.com/en/tutorials/video/basics/#toc-fun) at <http://www.html5rocks.com/en/tutorials/video/basics/#toc-fun>.

Audio element

Until recently, the ability to play any type of audio within a browser involved

using Adobe Flash or other browser plugins. Although Adobe's Flash player is unquestionably the most ubiquitous of these, most developers and designers would agree that it's better not to rely on a plugin at all. One of the most exciting and long-awaited features in HTML5 is the `<audio>` element, enabling native audio playback within the browser. The audio element APIs allows you to access control and manipulate timeline data and network states of the files.

The audio tag is supported by nearly all of the major browsers. For browsers that don't support audio natively you can always fallback to Flash.

The Spec

Currently, the HTML5 spec defines five attributes for the `<audio>` element:

src — a valid `<abbr>URL</abbr>` specifying the content source.

autoplay — a boolean specifying whether the file should play as soon as it can.

loop — a boolean specifying whether the file should be repeatedly played.

controls — a boolean specifying whether the browser should display its default media controls.

preload — none / metadata / auto — where 'metadata' means preload just the metadata and 'auto' leaves the browser to decide whether to preload the whole file.

Let's create a simple example that will play an audio file:

```
<audio src="elvis.ogg" controls preload="auto" auto-  
buffer></audio>
```

This table details the codecs supported by today's browsers:

To create your own controls, you can use the API methods defined by the spec:

- ▶ `play()` — plays the audio
- ▶ `pause()` — pauses the audio
- ▶ `canPlayType()` — interrogates the browser to establish whether the given mime type can be played
- ▶ `buffered()` — attribute that specifies the start and end time of the buffered part of the file

The source

The best way to force the browsers into playing audio is to use the `<source>` element. The browser will try to load the first audio source, and if it fails

or isn't supported, it will move on to the next audio source. In addition, we can embed a Flash player if all else fails:

- `<audio controls preload="auto" autobuffer>`
- `<source src="elvis.mp3" />`
- `<source src="elvis.ogg" />`
- `<!-- now include flash fall back -->`
- `</audio>`

Cross-Browser Implementation

The earlier audio players relied on Flash to implement the player's graphical interface, effectively isolating the player from the rest of the web design process. With growing support for HTML5 in modern browsers, developers are breaking their Flash dependency and now using native audio as it is supported.

One of the most significant issue is the cross-browser implementation, where lack of a common supported audio format among browsers causes complications. If developers want to take full advantage of all browsers that support HTML5 audio, they'll need to create both MP3 and Ogg (and in Opera's case, WAV) versions of the audio file they want to stream!

Since the HTML5 standard is still a work in progress, several aspects of the `<audio>` element vary from browser to browser. For example, there seems to be no way to determine the load progress of an audio file in all browsers as this relies on buffered DOM attribute being implemented. Chrome has implemented buffered while in Safari it's there but it behaves slightly differently. In order to compete effectively with plugin-based solutions, it is necessary that HTML5 audio implementation be consistent across all browsers and match current implementations feature for feature.

JavaScript solutions

In order to take advantage of each browser's audio capabilities, it is necessary to create different solutions for different browsers. You will need to check what capabilities a particular browser supports and adapt accordingly. Using JavaScript, you can check for audio tag support and file compatibility:

- `// returns a boolean`
- `var audioTagSupport= !! (document.createElement('audio').canPlayType);`
-
- `// Need to check the canPlayType first or an exception`
-

```

•   var myAudio = document.createElement('audio');
•
•   if (myAudio.canPlayType) {
•
•       // Currently canPlayType(type) returns: "", "maybe"
or "probably"
•
•       var canPlayMp3 = !!myAudio.canPlayType&& "" !=
myAudio.canPlayType('audio/mpeg');
•       var canPlayOgg = !!myAudio.canPlayType&& "" !=
myAudio.canPlayType('audio/ogg; codecs="vorbis"');
•   }

```

In order to change the src attribute of an audio object or element, you'll need to recreate the object or element with the new value for its src attribute or

Chrome	Firefox	Safari	Opera	IE	iOS	Android	Opera mobile
10+	3.5+	4+	10.5+	10+	4.1+	2.3+	11

Video element browser support

alternatively change the src URL and then issue a myAudio.load() command. So, to create a solution that takes full advantage of HTML5 audio, you'll typically need to:

1. Check for HTML5 audio support, and if not present, fall back on Flash,
2. Check the level of HTML5 audio support and adapt your code accordingly for each browser; and
3. Check what file types are supported and link to appropriate formats of the files.

Although HTML5 audio is a relatively new part of the standard, if recent trends continue and users upgrade to the latest versions of Safari, Firefox, Chrome and Opera, browser support is pegged at over 30 per cent. This is a significant chunk of the browser market that will no longer need to rely on browser plugin for audio support. When you consider that mobile and other lower-spec devices such as tablets are choosing to support HTML5 audio, you

Chrome	Firefox	Safari	Opera	IE	iOS	Android	Opera mobile
10+	3.5+	4+	10.5+	10+	4.1+	2.3+	11

Audio element browser support

begin to paint a picture of how important native audio support is becoming.

Canvas and Video

“Your powers combine... I am Captain Planet!”

Together? Yes! This wonderful combination opens up an entirely new dimension for web. On a canvas, you have control over every pixel of the canvas. So when a video runs on a canvas, you have entire control over the video! Real time modifications can be made to the video. Various new effects can be added to the video like changing backgrounds, manipulating colours, breaking it up, putting together and anything at all.

In the you can see clicking the video causes it to fragment into pieces, each fragment still playing a part of the video!

Geolocation

The Geolocation API lets you find out where the user is and keep tabs on them as they move around, always with the user’s consent. This functionality could be used as part of user queries, e.g. to guide someone to a destination point. It could also be used for “geo-tagging” some content the user has created, e.g. to mark where a photo was taken.

The API is device-agnostic; it doesn’t care how the browser determines location, so long as clients can request and receive location data in a standard way. The underlying mechanism might be via GPS, wifi, or simply asking the user to enter their location manually. Since any of these lookups is going to take some time, the API is asynchronous; you pass it a callback method whenever you request a location.

At it’s most simple use, the Geolocation API looks like this:

```
• function get_location() {
•   navigator.geolocation.getCurrentPosition(show_map);
• }
```

HTML5Rocks.com has a great sample application that shows how to calculate distance travelled between two points. Check out http://www.html5rocks.com/en/tutorials/geolocation/trip_meter/ for more details.

Web storage and offline web apps

“Web” and “online” are two closely associated terms, downright synonymous to many people. So why on earth would we talk about “offline” web technologies, and what does the term even mean? HTML5 and related

specs introduce a number of features to make offline web apps a reality like application cache, local storage, and indexedDB.

There are several reasons to use client-side storage. First, you can make your app work when the user is offline, possibly sync'ing data back once the network is connected again. Second, it's a performance booster; you can show a large corpus of data as soon as the user clicks on to your site, instead of waiting for it to download again. Third, it's an easier programming model, with no server infrastructure required. Of course, the data is more vulnerable and the user can't access it from multiple clients, so you should only use it for non-critical data, in particular cached versions of data that's also "in the cloud".

Session and Local Storage for web applications

Web Storage simply provides a key-value mapping, e.g. `localStorage["name"] = username;`. Unfortunately, present implementations only support string-to-string mappings, so you need to serialise and de-serialise other data structures. You can do so using `JSON.stringify()` and `JSON.parse()`.

There are two types of storage, `sessionStorage` and `localStorage`. Session Storage is primarily intended for scenarios where the user is carrying out a single transaction. Information that is put in the store is available across all windows tabs and frames for the entire session the browser stays open. Local Storage persists across sessions, so even after closing and reopening the browser, the data is still available.

Browser support for session and local storage is pretty good. All the new and popular browsers support it. You can use Modernizr to detect support for Web Storage.

```

• if (Modernizr.localstorage) {
•   // window.localStorage is available!
• } else {
•   // no native support for HTML5 storage :(
•   // maybe try dojox.storage or a third-party solution
• }

```

Setting up Web Storage

For `localStorage` or `sessionStorage`, we only need to check the store exists and is of the right type. If not, we'll create a new array and store it against the `localStorage` "checkins" key. We use JSON to convert the structure to a

string first, since, in most browsers, localStorage only stores strings.

```
• if (!localStorage["checkins"]) {
•   localStorage["checkins"] = JSON.stringify([]);
• }
```

Saving to Web Storage

With localStorage, we simply pull the checkins array out, add a new one to the end, and save it again. We also have to do the JSON dance to store it in string form.

```
• var checkins = JSON.parse(localStorage["checkins"]);
• checkins.push(checkin);
• localStorage["checkins"] = JSON.stringify(checkins);
```

Searching and Reading from Web Storage

The next function fishes out all checkins matching a particular mood, so the user can see where and when they were happy recently, for example. With localStorage, we have to manually walk through each checkin and compare it to the mood, building up a list of matches. It's good practice to return clones of the data that's stored, rather than the actual objects, since



Video courtesy: The Big Buck Bunny

Add new effects: As you can see, clicking the video causes it to fragment into pieces, each fragment still playing the video!

searching should be a read-only operation; hence we pass each matching checkin object through a generic clone() operation.

```

• var allCheckins = JSON.parse(localStorage["checkins"]);
• var matchingCheckins = [];
• allCheckins.forEach(function(checkin) {
•   if (checkin.mood == moodQuery) {
•     matchingCheckins.push(clone(checkin));
•   }
• });
• handler(matchingCheckins);

```

Providing Offline Experiences

It's becoming increasingly important for web-based applications to be accessible offline. Yes, all browsers have caching mechanisms, but they're unreliable and don't always work as you might expect. HTML5 addresses some of the annoyances of being offline with the ApplicationCache interface.

Using the cache interface gives your application three advantages:

- ▶ Offline browsing - users can navigate your full site when they're offline
- ▶ Speed - cached resources are local, and therefore load faster.
- ▶ Reduced server load - the browser will only download resources from the server that have changed.

The Application Cache (or AppCache) allows a developer to specify which files the browser should cache and make available to offline users. Your app will load and work correctly, even if the user presses the refresh button while they're offline.

Using Application Cache

To enable the application cache for an app, include the manifest attribute on the document's html tag:

```

• <html manifest="example.appcache">
•   ...
• </html>

```

The manifest attribute should be included on every page of your web application that you want cached. The browser does not cache a page if it does not contain the manifest attribute (unless it is explicitly listed in the manifest file itself. This means that any page the user navigates to that

include a manifest will be implicitly added to the application cache. Thus, there's no need to list every page in your manifest. A manifest file must be served with the mime-type text/cache-manifest. You may need to add a custom file type to your web server or .htaccess configuration.

Structure of a manifest file

A manifest can have three distinct sections: CACHE, NETWORK, and FALLBACK.

- ▶ **CACHE** - This is the default section for entries. Files listed under this header (or immediately after the CACHE MANIFEST) will be explicitly cached after they're downloaded for the first time.
- ▶ **NETWORK** - Files listed under this section are white-listed resources that require a connection to the server. All requests to these resources bypass the cache, even if the user is offline. Wildcards may be used.
- ▶ **FALLBACK** - An optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback. Both URIs must be relative and from the same origin as the manifest file. Wildcards may be used.

A simple manifest may look something like this:

CACHE MANIFEST

index.html

stylesheet.css

images/logo.png

scripts/main.js

- ▶ This example will cache four files on the page that specifies this manifest file.
- ▶ There are a couple of things to note:
- ▶ The CACHE MANIFEST string is the first line and is required.
- ▶ Sites are limited to 5MB worth of cached data. However, if you are writing an app for the Chrome Web Store, using the unlimitedStorage removes that restriction.
- ▶ If the manifest file or a resource specified in it fails to download, the entire cache update process fails. The browser will keep using the old application cache in the event of failure.


You must modify the manifest file itself to inform the browser to refresh cached files. Creating a comment line with a generated version number, hash of your files, or timestamp is one way to ensure users have the latest

Note

These sections can be listed in any order and each section can appear more than one in a single manifest.

version of your software. You can also programmatically update the cache once a new version is ready as described in the Application Cache tutorial on HTML5Rocks.com at <http://www.html5rocks.com/en/tutorials/appcache/beginner/>.

Closing Words

Open web technologies have a promising future enabling web developers to create experiences on the web that are more application like, and provide a richer, more immersive experience. No matter the form factor or device creating the next, great experience is in the hands of web developers building with HTML5, CSS3 and the rest of the open web platform. 

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



HOSTED APPS AND PACKAGED APPS

A guide to choosing and implementing the right kind of app

Google Chrome offers you the option of publishing your app in two ways – hosted or packaged. Sometimes it's hard to decide which one you need to publish. We'll try to highlight what characterises each of them and the various parallels, as well as the differences between them, so that you can make the right choice for your app.

What are apps?

In the context of Chrome and the Chrome Web Store, an app is an application that you run inside your browser with a dedicated user interface and user interaction. They're also known as installable web apps, and they're essentially a way of making Chrome treat certain web sites like apps.

As we are already aware by now, web apps combine the best aspects of a web site and a simple desktop application. A web app is much more rich and interactive than a web site, but less cumbersome and monolithic than a desktop application. Many installable web apps are hosted apps – normal web sites with a bit of extra metadata. You can build and deploy hosted apps exactly as you would build and deploy any web app, using any server-side or client-side technology you prefer. The only difference is that you must provide a small manifest file that describes the app. More on that later.

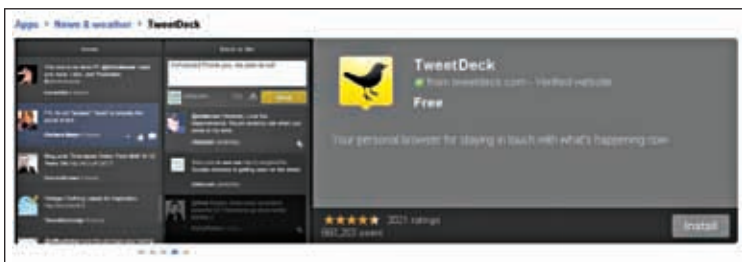
If you want your app to work especially well offline or to be tightly integrated with the Google Chrome browser, you can create a packaged app. A packaged app is just a web app that a user downloads. These apps have the option of using the Google Chrome Extension APIs, allowing them to change the way Chrome behaves or looks.

Hosted apps

If you already have a web app and are unwilling to tweak around with it, you can just publish it as a hosted app. A hosted app requires a CRX



Hosted App: Google Books is a hosted app



Packaged App: TweetDeck is a very popular packaged app

file that contains metadata describing the app. The CRX file for a hosted app must contain an icon and a manifest that has details about how the app should function. Do note, however, that a hosted app has no access to the files inside its CRX file.

Manifest

The manifest is a file named `manifest.json` that contains important information about the web app. We've covered more on Manifest in the next chapter. The following code is an example of a manifest:

```
• {
•   "name": "Great App Name",
•   "description": "Pithy description (132 characters or
less, no HTML)",
•   "version": "0.0.0.1",
•   "icons": {
•     "128": "icon_128.png"
•   },
•   "app": {
•     "urls": [
•       "http://mysubdomain.example.com/"
•     ],
•     "launch": {
•       "web_url": "http://mysubdomain.example.com/"
•     }
•   }
• }
```

For details about this example manifest, see the store's Getting Started tutorial: http://code.google.com/chrome/webstore/docs/get_started_simple.html

App icon

You must provide a 128x128-pixel app icon image in the ZIP file of your app. Some requirements for the image:

The actual icon size should be 96x96 (for square icons); an additional 16 pixels per side should be transparent padding, adding up to 128x128 total image size.

The image must be in PNG format.

To make sure your icon looks good, preview your app's listing in the

Chrome Web Store as well as in the New Tab page. For help on designing icons, see the “App icon” section of the Supplying Images page: <http://code.google.com/chrome/webstore/docs/images.html>.



Localization files

Unless your app is strictly local in scope, you should internationalize and localize it. Internationalization (or i18n) is the process of making it easy to adapt your app to various languages and regions. You can then localize your app—translate and otherwise adapt it so that it works well in a particular locale.

Even if you don't translate your app, you might want to display different information in the store for different locales. You can do this by modifying the manifest and providing localization files. Details are in the internationalization documentation:

<http://code.google.com/chrome/webstore/docs/i18n.html>

Auto-updating

It's obviously easy to update a hosted app—just update your web server—but it's equally easy to update a packaged app. You just upload the updated ZIP file for your app to the dashboard, and the store and Chrome take care of distributing the new version.

Background processing

Both hosted and packaged apps can perform actions in the background, using a background page or window. They can even request that Chrome start up early and not exit unless the user explicitly quits the browser.

The details of how you implement this capability depend on whether you are writing a hosted or packaged app. For more information, see: <http://code.google.com/chrome/apps/docs/background.html> (hosted apps) http://code.google.com/chrome/extensions/background_pages.html (packaged apps)

Choosing between hosted and packaged apps

This section gives some tips to help you decide which type of app is right for you.

If you already have a web app, you don't need to change it; you can just publish it as a hosted app. However, you might be interested in some of the

advantages that packaged apps provide, such as tight integration with the browser or the ability to download the entire app before ever running it.



Hosted App: Stay up to date with NYTimes

Hosted apps

- ▶ Hosted apps can usually work offline with the judicious use of Application Caching, an HTML5 technology.
- ▶ If your CRX file exceeds a size of 10 MB, a hosted app becomes a better choice.
- ▶ Use a hosted app if your existing web app uses server-side scripting to generate your pages and content, or if your app relies on form submissions and full page refreshes.
- ▶ Hosted apps will update automatically, just by updating your server.

Packaged apps

- ▶ Apart from ‘app’ behaviour, packaged apps also provide ‘extension’ behaviour. This can be exploited to improve the experience of your apps and differentiate from similar products.
- ▶ Because packaged apps are on the user’s local drive, they can launch immediately, without waiting for files from the web.
- ▶ Packaged apps can run in the background, even when the user has never launched your app.
- ▶ Apps that do not require a network connection, are ideal candidates for packaged apps because the user will be able to run them offline.

Additional info

If you are still not able to make up your mind, the following additional guidelines may do the trick for you:

- ▶ A hosted app can work in all web browsers, as long as it doesn't depend on some unique features of particular browsers. However, because only Chrome supports the Chrome Web Store, you have the option of tailoring your app to Chrome.
- ▶ A packaged app will work only in Chrome. However, you can reuse the app's code for ordinary web apps.
- ▶ The Licensing API lets you check programmatically whether the user has paid for your app using Chrome Web Store Payments. The Licensing API is usually used only by hosted apps because packaged apps are stored locally and could be modified to circumvent the API call.
- ▶ There are risks in selling a packaged app. Motivated people can bypass payments for packaged apps. An alternative to consider is moving to a hosted app model with a server-side licensing check.

Additional reading

Choosing an app type:

<http://code.google.com/chrome/webstore/docs/choosing.html>

Hosted app developer guide:

http://code.google.com/chrome/apps/docs/developers_guide.html

Packaged app developer guide:

<http://code.google.com/chrome/extensions/apps.html>

Internationalization guide:

<http://code.google.com/chrome/webstore/docs/i18n.html>

Manifests in hosted apps:

http://code.google.com/chrome/apps/docs/developers_guide.html#manifest

Manifest reference:

<http://code.google.com/chrome/extensions/manifest.html>

Chrome Web Store Licensing API:

http://code.google.com/chrome/webstore/docs/check_for_payment.html

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



UNDERSTANDING MANIFEST

A closer look at making your app exploit the special features of the Chrome Web Store by using the Manifest file

Any installable web app, extension or theme will have a JSON-formatted manifest file, named `manifest.json`, associated with it which contains important information.

Fields

Here is a sample code that shows the supported manifest fields. The only fields that are always required are name and version. They must be included in your manifest file. The others are not necessary, you may or may not use the depending on what your app does.

```
• {  
•   // Required  
•   "name": "My Extension",  
•   "version": "versionString",
```

```

• // Recommended
• "description": "A plain text description",
• "icons": { ... },
• "default_locale": "en",
•
• // Pick one (or none)
• "browser_action": {...},
• "page_action": {...},
• "theme": {...},
• "app": {...},
•
• // Add any of these that you need
• "background_page": "aFile.html",
• "chrome_url_overrides": {...},
• "content_scripts": [...],
• "homepage_url": "http://path/to/homepage",
• "incognito": "spanning" or "split",
• "key": "publicKey",
• "minimum_chrome_version": "versionString",
• "omnibox": { "keyword" : "aString" },
• "options_page": "aFile.html",
• "permissions": [...],
• "plugins": [...],
• "update_url": "http://path/to/updateInfo.xml"
• }

```

Fields Explained

A short description of the fields used above.

name

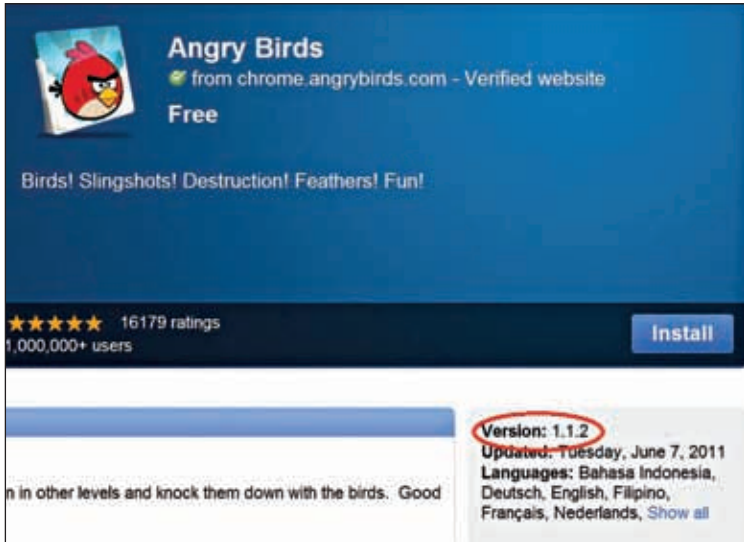
This unique plain text string (no more than 45 characters) will be your apps identification mark. This will be used in the install dialog, extension management UI, and the chrome web store. You can use locale-specific strings in this field

version

These one to four dot-separated integers identify the version of your app. A couple of rules apply to the integers: they must be between 0 and 65535, inclusive, and non-zero integers can't start with 0. For example, 99999 and 032 are both invalid.

Here are some examples of valid versions:

- ▶ “version”: “1”
- ▶ “version”: “1.1”
- ▶ “version”: “4.110.3”
- ▶ “version”: “2.1.3.33567”



The version is displayed in the right column

The autoupdate system compares versions to determine whether an installed extension needs to be updated. If the published extension has a newer version string than the installed extension, then the extension is automatically updated.

The comparison starts with the leftmost integers. If those integers are equal, the integers to the right are compared, and so on. For example, 1.2.0 is a newer version than 1.1.9.9999.

Eg. A missing integer is equal to zero. For example, 1.1.9.9999 is newer than 1.1.

description

This is a plain text string (no HTML or other formatting), no more than 132 characters, that describes the function and utility of your app. The description is

important because it will be used for both the browser's extension management UI and the Chrome Web Store. Locale-specific strings are accepted here too.

icons

This will be your identification mark on the web. You can use one or more icons to represent your extension, app, or theme. You should always provide a 128x128 icon; it's used during installation and by the Chrome Web Store. If you are making an extension, provide a 48x48 icon, which is used in the extensions management page (<chrome://extensions>). You can also specify a 16x16 icon to be used as the favicon for an extension's pages.

Although icons are preferred in PNG format, because of its support for transparency but you can be in any format supported by WebKit, including BMP, GIF, ICO, and JPEG. It is very important that you use only the documented icon sizes. This is because the details of Chrome's UI may change



Icons are the “face” of your app in the Web Store

between versions, and these changes assume that developers are using the documented sizes. If you use other sizes, your icon may look bad in future versions of the browser.

Here's an example of specifying the icons:

- `"icons": { "16": "icon16.png",`
- `"48": "icon48.png",`
- `"128": "icon128.png" },`

While uploading your extension, app, or theme using the Chrome Developer Dashboard, you will have to supply additional images including the screenshot of your app.

default_locale

Specifies the subdirectory of `_locales` that contains the default strings for this

extension. This field is required in extensions that have a `_locales` directory; it must be absent in extensions that have no `_locales` directory.

app

Used by installable web apps, including packaged apps, to specify the URLs that the app uses. Most important is the launch page for the app—the page that the browser goes to when the user clicks the app’s icon in the New Tab page.

For details, see the documentation for hosted apps and packaged apps.

default_locale

This field will specify the subdirectory of `_locales` that contains the default strings for your extension. You must include this field in extensions that have a `_locales` directory; it must be absent in extensions that have no `_locales` directory.

homepage_url

This is the URL of the homepage for your extension. The extensions management page (<chrome://extensions>) will contain a link to this URL. This comes in handy when you have to host the extension on your own site. If you distribute your extension using the Extensions Gallery or Chrome Web Store, the homepage URL defaults to the extension’s own page.

incognito

You can specify how your extension or web app will behave if it is run in the incognito mode.

The default mode for your extensions is “spanning”, which basically means that the extension will run in a single shared process. So messages from an incognito tab will be sent to the shared process, with an incognito flag indicating where it came from and processed there.

The default mode for installable web apps is “split”, which implies that all app pages in that incognito window will run in their own incognito process. Any background pages that your app contains will also run in the incognito process. Your regular process and the incognito process will not interfere or communicate with each other. The incognito process has a separate memory-only cookie store. You will only see the incognito tab updates in the incognito window.

While developing your app, if it needs to be logged into a remote server,

use the spanning incognito behavior. If you want to run your app independently in an incognito window, use the split behavior.

key

Your key controls the unique ID of an extension, app, or theme when it is loaded during development. Generally, you don't need to use this you can develop your code in such a way that this would not matter. Eg. using relative paths and `chrome.extension.getURL()`.

To get a suitable key value, first install your extension from a .crx file (you may need to upload your extension or package it manually). Then, in your user data directory, look in the file `Default/Extensions/<extensionId>/<versionString>/manifest.json`. You will see the key value filled in there.

minimum_chrome_version

This is used to specify the version of Chrome that your extension, app, or theme requires to run. The format for this string is the same as for the version field.

permissions

We have an idea of what permissions from a user's perspective. Now let's look at the developer side of permissions. Permissions will allow you to access a user's data which is critical to the function of any app. An app may use a single or an array of permissions depending on its requirements. Each permission can be either one of a list of known strings (such as "geolocation") or a match pattern that gives access to one or more hosts. Any permission that you use is displayed to the user before he installs the app.

Here's an example of the permissions part of a manifest file for an extension:

```
• "permissions": [
•   "geolocation",
•   "bookmarks",
•   "notifications",
•   "http://*.google.com/",
•   "unlimitedStorage"
• ],
```

The following table lists the permissions an extension or packaged app can use.

Notes

Hosted apps can use the "background", "geolocation", "notifications", and "unlimitedStorage" permissions, but not any other permissions listed in this table.

PERMISSION	DESCRIPTION
match pattern	Specifies a host permission. Required if the extension wants to interact with the code running on pages. Many extension capabilities, such as cross-origin XMLHttpRequests, programmatically injected content scripts, and the cookies API require host permissions. For details on the syntax, see Match Patterns.
"background"	Makes Chrome start up early and shut down late, so that apps and extensions can have a longer life. When any installed hosted app, packaged app, or extension has "background" permission, Chrome runs (invisibly) as soon as the user logs into their computer—before the user launches Chrome. The "background" permission also makes Chrome continue running (even after its last window is closed) until the user explicitly quits Chrome. Note: Disabled apps and extensions are treated as if they aren't installed. You typically use the "background" permission with a background page or (for hosted apps) a background window.
"bookmarks"	Required if the extension uses the chrome.bookmarks module.
"chrome://favicon/"	Required if the extension uses the "chrome://favicon/url" mechanism to display the favicon of a page. For example, to display the favicon of http://www.google.com/ , you declare the "chrome://favicon/" permission and use HTML code like this: <code></code>
"contextMenus"	Required if the extension uses the chrome.contextMenus module.
"cookies"	Required if the extension uses the chrome.cookies module.
"experimental"	Required if the extension uses any chrome.experimental.* APIs.
"geolocation"	Allows the extension to use the proposed HTML5 geolocation API without prompting the user for permission.
"history"	Required if the extension uses the chrome.history module.
"idle"	Required if the extension uses the chrome.idle module.
"management"	Required if the extension uses the chrome.management module.
"notifications"	Allows the extension to use the proposed HTML5 notification API without calling permission methods (such as checkPermission()). For more information see Desktop Notifications.
"tabs"	Required if the extension uses the chrome.tabs or chrome.windows module.
"unlimitedStorage"	Provides an unlimited quota for storing HTML5 client-side data, such as databases and local storage files. Without this permission, the extension is limited to 5 MB of local storage. Note: This permission applies only to Web SQL Database and application cache (see issue 58985). Also, it doesn't currently work with wildcard subdomains such as http://*.example.com .

browser action

browser actions can be used to put an icon, tooltip, badge and pop ups in Google Chrome. Packaged apps cannot use browser actions.

A browser action in your manifest will look something like this:

```
• {
•   "name": "cool app name",
•   ...
•   "browser_action": {
•     "default_icon": "images/image001.png", // optional
•     "default_title": "ThinkDigit",        // tooltip,
optional
•     "default_popup": "popup.html"         // optional
•   },
•   ...
• }
```

Icon: icons can be up to 19 pixels wide and high. Any larger icons will be resized to fit, so use the mentioned icon size for best results. Images can be in BMP, GIF, ICO, JPEG, or PNG i.e. any format WebKit can display. You can have animated UIs using the canvas element.

To set the icon, use the `default_icon` field of `browser_action` in the manifest, or call the `setIcon()` method.

`chrome.browserAction.setIcon(object details)`

Tooltip: to set the tooltip, use the `default_title` field of `browser_action` in the manifest, or call the `setTitle()` method. You can specify locale-specific strings for the `default_title` field; see Internationalization for details.

Badge: you can choose your browser action to optionally display a badge which is a bit of text that is layered over the icon. Badges make it easy to update the browser action to display a small amount of information about the state of the extension. You should display the information in 4 characters or less because of the limited space if the badge.

Set the text and color of the badge using `setBadgeText()` and `setBadgeBackgroundColor()`, respectively.

Popup: Including this browser action in your manifest would make a popup appear when the user clicks the icon. This new popup can contain any HTML contents that you like which will be automatically sized to fit its contents.

To add a popup to your browser action, create an HTML file with the popup's contents. Specify the HTML file in the `default_popup` field of

browser_action in the manifest, or call the setPopup() method.

For the best visual impact, follow these guidelines:

- ▶ Use browser actions for features only when necessary. If they make sense only for few pages, use page action instead
- ▶ Do use big, colourful icons that make the most of the 19x19-pixel space. Browser action icons should seem a little bigger and heavier than page action icons.
- ▶ Soften the edges of your icon as it goes better with different themes that people use.
- ▶ Animated icons are annoying.

Page actions

To put icons inside the address bar, you need page actions. Page actions represent actions that can be taken on the current page, but that aren't applicable to all pages. For example, subscribing to RSS feeds or making a collage from the photos on the page.



Subscribe to RSS feed with a single click

To make your

extensions icon always visible, you should use browser actions.

After including page action in your manifest, it should look something like this:

```

• {
•   "name": "My extension",
•   ...
•   "page_action": {
•     "default_icon": "icons/foo.png", // optional
•     "default_title": "Do action",    // optional; shown
in tooltip
•     "default_popup": "popup.html"   // optional
•   },
•   ...
• }

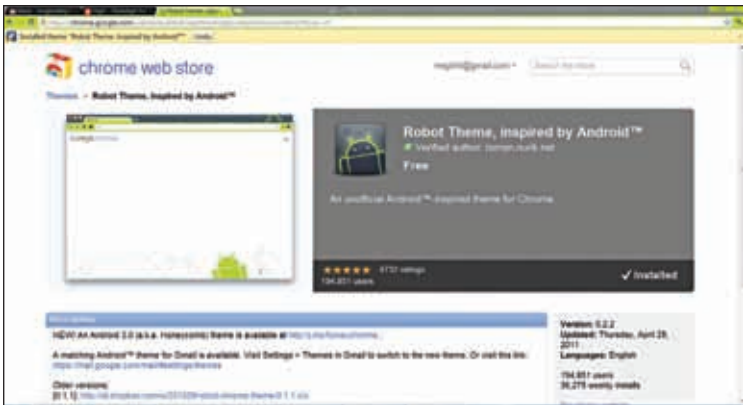
```

Similar to browser actions, page actions can have an icon, a tooltip, and popup but they can't have badges. Page actions are not always visible, you can configure them to manage their appearance.

You make a page action appear and disappear using the show() and hide() methods, respectively. By default, a page action is hidden. When you show it, you specify the tab in which the icon should appear. The icon remains visible until the tab is closed or starts displaying a different URL. Packaged apps cannot use page actions.

Themes

A theme is just like an extension which changes the way your browser looks. They don't have any function besides 'looking cool'.



Android theme applied to Google Chrome

This is a manifest file for a theme:

```
{
  "version": "2.6",
  "name": "camo theme",
  "theme": {
    "images" : {
      "theme_frame" : "images/theme_frame_camo.png",
      "theme_frame_overlay" : "images/theme_frame_stripe.png",
      "theme_toolbar" : "images/theme_toolbar_camo.png",
      "theme_ntp_background" : "images/theme_ntp_background_norepeat.png",
      "theme_ntp_attribution" : "images/attribution.png"
```

```

•   },
•   "colors" : {
•     "frame" : [71, 105, 91],
•     "toolbar" : [207, 221, 192],
•     "ntp_text" : [20, 40, 0],
•     "ntp_link" : [36, 70, 0],
•     "ntp_section" : [207, 221, 192],
•     "button_background" : [255, 255, 255]
•   },
•   "tints" : {
•     "buttons" : [0.33, 0.5, 0.47]
•   },
•   "properties" : {
•     "ntp_background_alignment" : "bottom"
•   }
• }
• }

```

Colours: Colours are in the standard RGB format. To find the strings you can use within the “colors” field, look for `kColor*` strings in `theme_service.cc`.

Images: Image resources use paths relative to the root of the extension.

Properties: This field lets you specify properties such as background alignment, background repeat, and an alternate logo.

Tints: You can specify tints to be applied to parts of the UI such as buttons, the frame, and the background tab. As of now, Google Chrome supports tints, not images, because images don’t work across platforms.

Tints are in Hue-Saturation-Lightness (HSL) format, using floating-point numbers in the range 0 - 1.0. Hue is an absolute value, with 0 and 1 being red. Saturation is relative to the currently provided image. 0.5 is no change, 0 is totally desaturated, and 1 is full saturation. Lightness is also relative, with 0.5 being no change, 0 as all pixels black, and 1 as all pixels white.

You can alternatively use -1.0 for any of the HSL values to specify no change.

Background page

You often have the need to have a single long-running script to manage some task or state. This is where background page comes to the rescue. This HTML page runs in the extension process. It runs as long as the app is running and only one instance is active at any given time.

In a typical extension with a background page, the UI — for example, the browser action or page action and any options page — is implemented by dumb views. When the view needs some state, it requests the state from the background page. When the background page notices a state change, the background page tells the views to update.

The background page looks something like this in your manifest:

```
• {
•   "name": "extension name",
•   ...
•   "background _ page": "background _ page.html",
•   ...
• }
```

To speed up the browser startup, consider specifying the background permission.

You can communicate between your various pages using direct script calls, similar to how frames can communicate. The `chrome.extension.getViews()` method returns a list of window objects for every active page belonging to your extension, and the `chrome.extension.getBackgroundPage()` method returns the background page.

The following code snippet demonstrates how the background page can interact with other pages in the extension. It also shows how you can use the background page to handle events such as user clicks.

The extension in this example has a background page and multiple pages created (with `chrome.tabs.create()`) from a file named `image.html`.

```
• //In the background page:
• <html>
•   <script>
•     //React when a browser action's icon is clicked.
•     chrome.browserAction.onClicked.
addListener(function(tab) {
•     var viewTabUrl = chrome.extension.getURL('image.
html');
•     var imageUrl = /* an image's URL */;
•
•     //Look through all the pages in this extension to
find one we can use.
•     var views = chrome.extension.getViews();
•     for (var i = 0; i < views.length; i++) {
•       var view = views[i];
```



```

•
•      //If this view has the right URL and hasn't been
used yet...
•      if (view.location.href == viewTabUrl && !view.
imageAlreadySet) {
•
•          //...call one of its functions and set a prop-
erty.
•          view.setImageUrl(imageUrl);
•          view.imageAlreadySet = true;
•          break; //we're done
•      }
•  }
•  });
•  </script>
• </html>
•
• //In image.html:
• <html>
•   <script>
•     function setImageUrl(url) {
•       document.getElementById('target').src = url;
•     }
•   </script>
•
•   <body>
•     <p>
•       Image here:
•     </p>
•
•     
•
•   </body>
• </html>

```

Override Pages

Using override pages you can substitute an HTML(contains CSS and JavaScript) file from your extension for a default page that Google Chrome offers.

This allows to replace any one of the following pages:

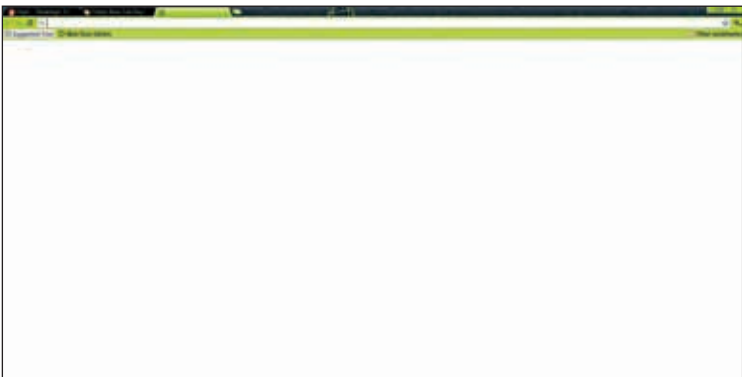
- ▶ **Bookmark Manager:** The page that appears when the user chooses the

Bookmark Manager menu item from the wrench menu or, on Mac, the Bookmark Manager item from the Bookmarks menu. You can also get to this page by entering the URL `chrome://bookmarks`.

- ▶ **History:** The page that appears when the user chooses the History menu item from the Tools menu or, on Mac, the Show Full History item from the History menu. You can also get to this page by entering the URL `chrome://history`.
 - ▶ **New Tab:** The page that appears when the user creates a new tab or window. You can also get to this page by entering the URL `chrome://newtab`.
- You can only override one of them



Default new tab



Blank new tab after installing empty new tab page

Since incognito windows run independently, you cannot override the New Tab pages cannot be overridden in incognito windows. You can override other pages in the incognito window by setting the incognito manifest property to spanning..

Override your new page

- `{// substitute the new page`
- `"name": "cool extension",`
- `...`
- `"chrome_url_overrides" : {`
- `"newpage": "myPage.html"`
- `},`
- `...`
- `}`

You can also have bookmarks or history in place of newpage.

While designing override pages, remember 2 things

Quick load: No user likes to wait, that too for a built in browser pages.

Do not set up any process that would take time loading

Title: The user can get confused by seeing the URL of your page. Keep it simple and clear.

NPAPI plugins

Warning: only use this when every other way does not work.

HTML and JavaScript are a great help in developing new extensions, but what if you wanted you use an existing code? Bundling an NPAPI plugin with your extension will allow you to call into native binary code from JavaScript. The administrator runs the binary code, so it runs with administrator privileges.

Any code that runs in an NPAPI plugin has the full permissions of the current user and so he is exposed to malicious input since he is no more shielded by Google Chrome in any way. You should be especially cautious when processing input from untrusted sources, such as when working with content scripts or XML[HttpRequest](#).

Extensions that use the NPAPI plugin will be manually reviewed before they can be published

After you've got an NPAPI plugin (See Mozilla's NPAPI plugin reference for information on how to do that), follow these steps to get your extension using it.

Add a section to your extension's `manifest.json` that describes where to find the plugin, along with other properties about it:

```
• {
•   "name": "My extension",
•   ...
•   "plugins": [
•     { "path": "content_plugin.dll", "public": true },
•     { "path": "extension_plugin.dll" }
•   ],
•   ...
• }
```

The “path” property specifies the path to your plugin, relative to the manifest file. The “public” property specifies whether your plugin can be accessed by regular web pages; the default is false, meaning only your extension can load the plugin.

Create an HTML file that loads your plugin by mime-type. Assuming your mime-type is “application/x-my-extension”:

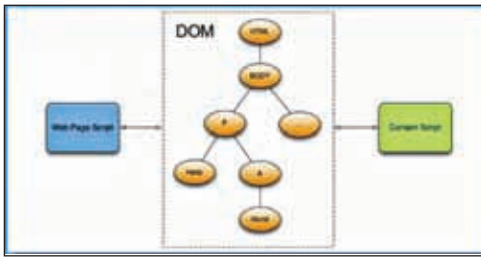
```
• <embed type="application/x-my-extension" id="pluginId">
• <script>
•   var plugin = document.getElementById("pluginId");
•   var result = plugin.myPluginMethod(); // call a method
in your plugin
•   console.log("my plugin returned: " + result);
• </script>
```

This can be inside a background page or any other HTML page used by your extension. If your plugin is “public”, you can even use a content script to programmatically insert your plugin into a web page.

Once you have an NPAPI plugin in your extension, the extension has unrestricted access to the local machine which is a security risk. If your plugin is not secure enough, a hacker could find loopholes and install potentially dangerous software on the user’s computer. With so many security issues its best to avoid including an NPAPI plugin whenever possible.

Content Scripts

Javascripts are used to carry out various function that are needed to perform different actions on the page ranging from trivial operations like updating fields to accessing and painting some animation on the canvas. Scripts are thus a means by which we can change most of the content on the page



DOM shared between the webpage scripts and the content scripts

according to user actions like clicks or key presses and browser actions like loading a page, closing a page etc.

When we use extensions over some webpage, the entities of the page may be changed in which they are presented before the user. Content scripts are basically javascripts that run operate upon the content in the web page in current context. But, the page might have its own javascripts that perform function that the site provides. How does it work out then?

The DOM (Document Object Model) is shared between the two scripts and both the scripts can operate upon the DOM independently.

With this ability given to the content scripts, they can be used to perform tasks like resizing text, changing element attributes, linking textual hyperlinks that are unlinked, downloading all media files that are present in the page and anything at all that can be done with the DOM. Changes made by any of the two scripts viz. the web page scripts and the content scripts to the DOM are visible to each of them.

Important point to be noted here is that the scripts should remain isolated from each other. They should not interfere with each other's functionality. The scripts may also use different versions of a given library, care is taken that they do not interfere with each other in any way. Isolation provided also takes care that the webpage scripts do not get access to the privileged API that the extensions are capable of using. Hence the data accessible through that API and sensitive user credentials are prevented from being leaked. All the content scripts are isolated from the webpage scripts and also from the other content scripts for different extensions.

Injecting your content scripts

The content scripts that needed to be injected as routine into the code should be registered in the manifest.

```

• {
•   "name": "My extension",
•   ...
•   "content_scripts": [
•     {
•       "matches": ["http://www.google.com/*"],
•       "css": ["mystyles.css"],
•       "js": ["jquery.js", "myscript.js"]
•     }
•   ],
•   ...
• }

```

If the content scripts are to be injected sometimes only then they are registered using the permissions filed.

```

• {
•   "name": "My extension",
•   ...
•   "permissions": [
•     "tabs", "http://www.google.com/*"
•   ],
•   ...
• }

```

Conditional injection

To specify conditional injections of context fields into a page, we can their URI patterns to distinguish them. If the URI matches pattern in the `include_globs` field, it will be included. If it matches pattern in `exclude_globs` field, it will be excluded.

For example, the glob `"http://???.random*.com/foo/*"` matches any of the following:

`"http://www.randompage.com/foo/bar"`

`"http://xyz.randomwebsite.com/foo/"`

Running somewhere on Mars! Isolated World

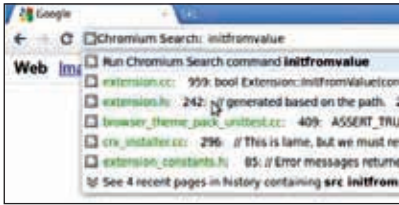
Isolation issue was mentioned briefly in the beginning of this topic. The isolation to the scripts is provided by an environment called as the isolated world. The scripts have access only to the DOM elements of the page. Access to local JS variables of the webpage scripts is disallowed. The visibility is limited and the environment is such that it seems that there is no other

javascript is executing on the current page. The reverse should also apply to maintain isolation. Hence the webpage javascripts are totally unaware of any other content scripts running on the page.

That's precisely most of what we need to know about content scripts and their interaction with page under operation.

Omnibox

Omnibox has been a feature of the chrome browser that has been much advertised. The address bar in chrome that doubles up for your search engine queries is known as the omnibox. Operations relevant to the omnibox



Keywords registered with the omnibox show up your extension

can be performed using the omnibox API. It allows you to register a keyword with Google Chrome's address bar, which is also known as the omnibox.

You must include an omnibox keyword field in the manifest to use the omnibox API. You should also specify a 16x16-pixel icon,

which will be displayed in the address bar when suggesting that users enter keyword mode.

For example:

```
{
  "name": "Aaron's omnibox extension",
  "version": "1.0",
  "omnibox": { "keyword" : "aaron" },
  "icons": {
    "16": "16-full-color.png"
  },
  "background _ page": "background.html"
}
```

The API is simple to use and is well documented at <http://code.google.com/chrome/extensions/omnibox.html>

Events like `onInputCancelled`, `onInputChanged`, `onInputEntered` can be used to trigger actions of different types. Suggestions can be made by calls like `setDefaultSuggestion`. In a nutshell, all operations we regularly see inside the omnibox can be programmed using the omnibox API.

Options

It is a desirable feature to be able to configure the behaviour of our extension by providing the user with options or preferences for the extension. Using the options field in the manifest can be used to provide a link to the page dedicated for personalizing the extension by changing its working options. If you do, a link to it will be provided from the extensions management page at <chrome://extensions>. Clicking the Options link opens a new tab pointing at your options page.

```

• {
•   "name": "My extension",
•   ...
•   "options_page": "options.html",
•   ...
• }
•

```

Default CSS styles may be provided in future by the Web Store for the options page to maintain a consistent look across different pages. 

SCAN THE **QR CODE** OR GO TO
[BIT.LY/CHROMECHALLENGE](https://bit.ly/chromechallenge)
 AND STAND A CHANCE TO WIN
 SAMSUNG GALAXY TABS



UPLOADING AND PUBLISHING

Now that you've prepared the manifest, we show you how to pack your apps and get them up and running on the Chrome Web Store!

On to the shelf finally!

Developing some wonderful app is the interesting part that creative people can do extremely well. But there are some rules of the game that everyone has to follow. There is a specific method (or process) by which you should upload your Web Apps into the open market. Uploading involves filling in lots of details about your app so that it falls under the right category in the store. Categorizing is an important aspect as it takes you to the customer. A telescope kept on sale in the fish market may reside there forever!

All the essential should be uploaded as a part of the extension so that every single aspect ranging from functionality to icons to favicons must work perfectly. Bugs should be properly removed and integrity of the application should be verified thoroughly. Every miniscule factor decides how well your app performs.

Pricing strategically is another aspect that decides the popularity of your web app and also balances the load of the efforts that you are undertaking to keep it a hot-pick in the store.

In this section, we will read about the method and best practices to be followed by developers for uploading and publishing their apps and extension.

My developer self...

Now that you have built your web app, it's time to unleash it before the world. To publish apps on the Chrome Web Store, you will first need to set your developer account. You can use any of your Google accounts for this purpose, but it is a good idea to have an account different from your personal account, dedicated for development purpose. If you're working on a hosted app, you may need to verify that the developer account owns the URLs comprising the Web App. Ownership can be verified using Google Webmaster Tools.

Uploading the app

After setting up the developer account, you will have to create a ZIP file of the folder containing your app. This ZIP file is to be uploaded on the Developer dashboard. It should include atleast include one file: the app's manifest. One or more icons that represent the extension, app, or theme should be provided. You should always provide a 128x128 icon; it's used during installation and by the Chrome Web Store. Extension management page uses a smaller icon of dimensions 48x48. Infobars use even smaller icon of 16x16. When the user leaves the page for which the infobar is displayed, the infobar automatically disappears. The small 16x16 icon is also used as a favicon for the extension management page. The point here being careful about including all essential files correctly when you are uploading the app.

Whereabouts

Once the upload is complete, details regarding the app are to be filled in stating its category, language, appearance, additional screenshots, Google-Analytics tracking ID if any, additional links or sources of information about the product. Depending upon the information you've entered, your app will be listed in the store under an appropriate category. This selection should be done carefully to get better user ratings for the app. Adding a YouTube video or a Google docs presentations is desirable as it will definitely create a good impression on the user and also provide him with help on its usage.

Icon 128x128px [Upload new image](#)

Appearance of header

☒ Use default background

☐ Custom image - 570x275px: [Upload new image](#)

See examples and learn more: [Tips and guidelines for images](#)

Upload screenshots:
400x275 pixels or proportionally larger (GIF, JPEG, or PNG)

[Choose File](#) No file chosen

Link to YouTube video or Google Docs Presentation (optional)

[Add](#)

App appearance and information resources

Trust issues

Another important thing required is that you verify the ownership of the website on which the app is hosted using Google's Webmaster Tools. If you have no previously verified websites, just follow the simple instructions after adding a new website to be verified.

App constants

To identify your web app, the Web Store provides you with an App ID. You

Google Analytics tracking

→ Your Google Analytics ID: UA- (example: 1234567-1) [Learn more about Google Analytics tracking](#)

Links

→ Link to website for your extension (optional)

→ Link to support & FAQ for your extension (optional)

App resource links and statistical data gathering

Verified website

This is an official item for a website I own:

[Refresh list](#)

→ [Add a new site with Google Webmaster Tools...](#)

Adding an official website or resource for the app

can get it from the URL of dashboard page for your app. For example, the URL <https://chrome.google.com/extensions/detail/sampleapp1piyush?hl=en> has the app ID sampleapp1piyush. The app ID finds its use whilst using the Licensing API, which allows the developer to correctly identify whether the user is using the paid version of the app or the free version. Depending on that features may be restricted. The app ID is also obtained when you receive the OAuth token. The OAuth access token and the access token secret is required when you need to implement payment gateways options.

Finishing touches...

After you are ready with all the stuff, it's time to finish off the process by adding any code that refers to the app ID or the OAuth access token. And we're almost done! You, the developer, can update the app as many times as required. With every new release all you need to do apart from updating the code is increase the version number. The auto update notifications in Chrome will automatically notify the user of any new version of your app that are available. Keeping your app frequently updated is a good idea because it is a motivating factor for the user to purchase the paid version of your app.

Lifetime validity recharge!

Before you can start publishing, you are required to register by paying a developer signup fee. This is a one time fee of \$5 that has to be paid before publishing your first app. Once registered, from next time on you are free to publish more apps without any fees.

...and finally

All said and done, we are now ready to open up the app before the entire world. Are we really ready? Bugs are undesirable and will earn you lots of bad ratings and make your app notorious. It's a nice idea to first open up the app only for some trusted testers who would help you find those hidden bugs. On the edit page for you app, there is a "Publish to test accounts" button that allows you to publish your app to selected few trust worthy pals.

You can add and remove tester accounts which will allow only the testers to view the listing in the store when they are logged in. One thing to be noted is that the app won't be a part of the search results yet, so you need to give the corresponding link to your app to the testers. For apps that are paid, it is advised that the first use test accounts to buy the app and then after logging into the app verify how the licensing server integrates with your app.

With the testing done properly now we are ready to get out of the nest and fly out into the world with all our might. This is a easy job; all you need to do is click the “Publish” link in your dash board beside your app.

The Chrome Web Store provides you numerous publishing options like selecting supported countries, languages, providing different pricing in different countries etc. To ensure maximum exposure to your app, you should internationalize it by providing support for various locales. Your app can be updated as and when required, so can the localization. So you may internationalize the app since the beginning and later on keep adding support for newer languages. Packaging your apps

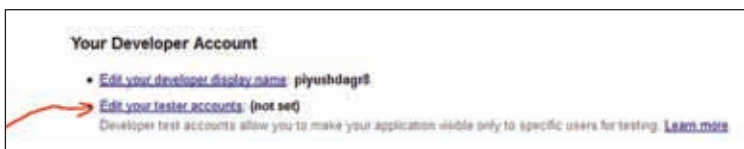
To begin with, packaging is creating bundled web app into a .crx file that can use Chrome extension features Sometime you may wish to give away your app to alpha tester without releasing a public version of the app, or you may not want to run a service to host your app or build offline installable



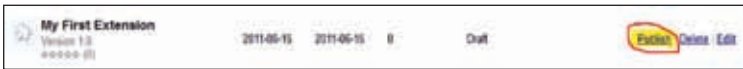
Testing before publishing through tester accounts

apps with good and rigid integration with the Chrome browser. Packaging can be useful for the purpose. But for any other purpose, there may not be any need to package your app when you are using the Chrome Developer Dashboard. Creating packaged apps is no different from creating any other extension, only thing is that they cannot include browser actions or page actions, so you cannot use those fields in the manifest. It includes atleast one HTML file that render the UI of the app.

CRX files are ZIP files with a special header and the .crx file extension. The header contains the author's public key and the extension's signature.



Modify the tester accounts



Publish: Just one click is all it takes

The signature is generated from the ZIP file using SHA-1 with the author's private key. The author need not worry too much about these details as the process almost completely automatic.

Let's quickly see how to package your apps.

First, go to the extensions page by typing in `chrome://extensions` in the address bar or by going to Tools > Extensions.

Unfold the "Developer Mode" by clicking the + if you see a –

Click the "Package Extension" button


In the window that pops up, in the Extension root directory field, give the path of the folder where the extension files are stored. Keep the next field ie. the Private Key file field blank. The key will be generated automatically.

After clicking OK, two files will be generated; A CRX file and a PEM file. The CRX file is the extension and the PEM file is the private key by which the package is signed. Important thing to remember is that you'll have to keep the private key safe. It will be required when you either update the package or when you wish to upload the extension using your dashboard.

Command line may be the love of many programmers, for them a single command can package the extension for you. Just hit in

```
• chrome.exe --package-extension=<PATH> --package-extension-key=<KEY PATH> [--no-message-box]
```

The optional argument above prevents the message box being displayed. And done!

In the future, you might (and should) update your extension, by making changes to the extension files and updating the version number in the manifest. You'll also need to update the package. The method is the same, only difference being, you'll have to provide the PEM private key file in the field which you previously had kept blank. 

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



BEST APPS

A collection of must have apps for everyone.

mflow

mflow makes it easy to find, stream and share new music without any annoying ads. With over 5 million free tracks, mflow makes sure that you will never run out of music.



Stream music without ads

Springpad

Springpad takes making and saving notes to a whole new level. You can save tasks, bookmarks, products, movies, places and anything you like for later. It automatically organises, and synchronises them, and adds important information and links. You can even set reminders that can be tweeted to you. You can access and edit information when you are offline.

Organise
yourself



TweetDeck

This is one of the most widely used twitter apps other than Hootsuite and Yoono. If you are using the desktop app, the web app will import all your data with a single click. As the developer puts it, "TweetDeck is your personal browser connecting you with your contacts across Twitter, Facebook, Foursquare and Google Buzz"



Stay con-
nected

SlideRocket

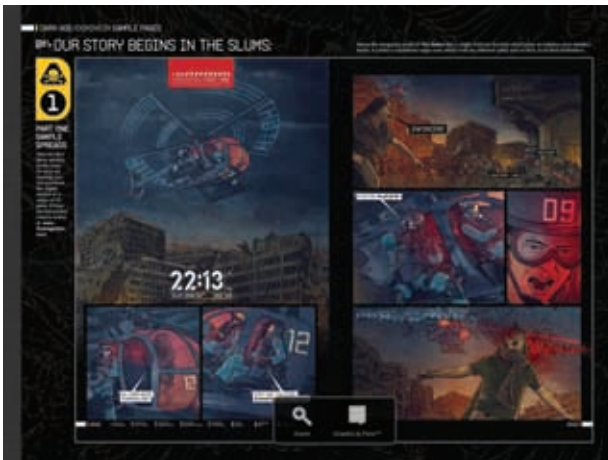
This is perhaps the only web app that performs comparable to what it was inspired from: Powerpoint. A great app to make and share presentations online.

Making presentations was never this fun



Graphic.ly

A must have app for every comic fan. Graphic.ly is a platform for digital comics and offers a lot more than just reading them. There are a lot of free comics but you have to shell out to read some of them.



No one is too old to read comics

Ebuddy

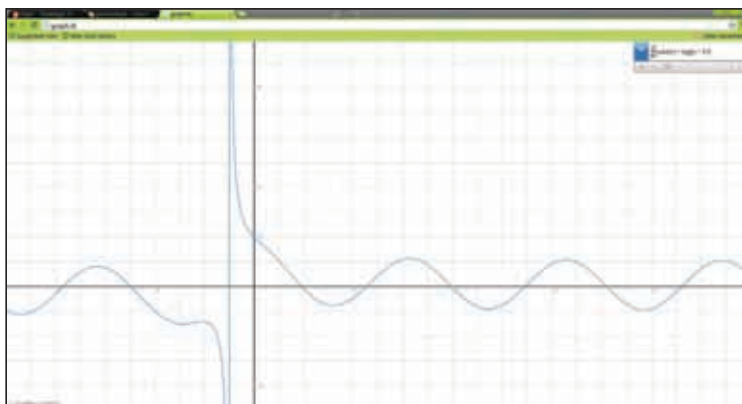
Now you can use your favourite messenger in the Chrome Web Store.



One messenger to chat with them all

Graph.tk

A must have for all students and analysts out there. A free, simple and powerful graphic calculator useful for plotting graphs



Just enter the function you want and it does the rest


Wikihow

This app teaches you the most important thing of all-survival. Right from doing extreme things like defending a tiger attack to simple things like changing your tire. It also has comprehensive articles on CPR, choking, natural disasters which can also be accessed online. You will find a way out of every situation that you can ever land up in.



This is the key to
your survival

Todo.ly

Todo.ly is a great app to create your todo lists for everyday work, ranging from business meetings to changing the batteries of your remote. You can set reminders and assign due dates to tasks and the app will automatically notify you as the deadlines come closer. 



You will never
miss anything
again

SCAN THE **QR CODE** OR GO TO
BIT.LY/CHROMECHALLENGE
AND STAND A CHANCE TO WIN
SAMSUNG GALAXY TABS



YOUR FIRST APP

Here we present the source code along with an explanation on writing a basic extension without any fancy features.

This is a sample program source code that demonstrates use of Chrome Web Store Licensing API in PHP. The program logs in with OpenID login and fetches the membership type of the user with the OAuth token and identifies the user either as a trial user, full access user or a no access user.

The code also uses pop ui extension to OpenID protocol. The user continues to stay on the same page and gets a popup login window. This avoids the user being redirected to the Google login page.

For detailed information go to <http://code.google.com/chrome/webstore/docs/samples.html>

```

• /* *
•  * Copyright 2010 the Chromium Authors
•  *
•  * Use of this source code is governed by a BSD-style
license that can be found
•  * in the "LICENSE" file.
•  *
•  * Eric Bidelman<ericbidelman@chromium.org>
•  */

```

Starting the session for the new user.

```

• session_start();
• require_once 'lib/oauth/OAuth.php';
• require_once 'lib/lightopenid/openid.php';

```

Here, “lib” contains the necessary OAuth and OpenID libraries to talk to the Chrome Web Store Licensing API and Google’s OpenID endpoint.

```

• // Full URL of the current application is running under.
• $scheme = (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS']
!= 'on') ? 'http' :
•
• 'https';
• $selfUrl = "$scheme://{$_SERVER['HTTP_HOST']}{$_
SERVER['PHP_SELF']}";
• /**
•  * Wrapper class to make calls to the Chrome Web Store
License Server.
•  */
• classLicenseServerClient {
•
•
• const LICENSE_SERVER_HOST = 'https://www.googleapis.com';
• const CONSUMER_KEY = 'anonymous';
• const CONSUMER_SECRET = 'anonymous';
• const APP_ID = '1'; // Change to the correct id of your
application.
• const TOKEN = '[REPLACE THIS WITH YOUR OAUTH TOKEN]';
• const TOKEN_SECRET = '[REPLACE THIS WITH YOUR OAUTH
TOKEN SECRET]';
• public $consumer;
• public $token;
• public $signatureMethod;
• public function __construct() {

```

```

• $this->consumer = new OAuthConsumer(
•     self::CONSUMER_KEY, self::CONSUMER_SECRET,
NULL);
• $this->token = new OAuthToken(self::TOKEN,
self::TOKEN_SECRET);
• $this->signatureMethod = new OAuthSignatureMethod_
HMAC_SHA1();
• }

```

\$url stores the complete URL that points to the resource that needs to be accessed.

\$request is used to contain the signed request that is to be made to the resource.

\$extraHeadervariable stores any additional headers that might be required.

\$returnResponseHeaders is set to True if the response headers are to be returned, else it is False.

```

• protected function send_signed_get($request,
$extraHeaders=NULL,
•
$returnRequestHeaders=false,
•
$returnResponseHeaders=false) {
•     $url = explode('?', $request->to_url());

```

Curl is a tool to transfer data to or from server, similar to wget. It supports a wide range of protocols. In the code below, the curl library customises various options of the connection.

```

• $curl = curl_init($url[0]); // starting a curl session to
create a url
• curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
• curl_setopt($curl, CURLOPT_FAILONERROR, false);
• curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);

```

The request headers here are sent back in the response

```

• curl_setopt($curl, CURLINFO_HEADER_OUT, $returnRe-
questHeaders);

```

As mentioned above, if we want the response headers, this property can be changed.

```

• if ($returnResponseHeaders) {
•   curl_setopt($curl, CURLOPT_HEADER, true);
• }

```

Any extra headers if present are appended to the headers

```

•   $headers = array($request->to_header());
• if (is_array($extraHeaders)) {
•   $headers = array_merge($headers, $extraHeaders);
• }
• curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);

```

After setting all the options like the destination url, the appropriate headers, we now send the request to the authentication server by issuing `curl_exec()`. In case of a response, it is stored.

```

•   // Execute the request. If an error occurs fill the
response body with it.
•   $response = curl_exec($curl);
• if (!$response) {
•   $response = curl_error($curl);
• }
•
•   // Add server's response headers to our response body
•   $response = curl_getinfo($curl, CURLINFO_HEADER_OUT)
. $response;
• curl_close($curl);
•
• return $response;
• }

```

The response returned may be either the desired one or an error message, depending upon the request sent to the authentication server resource.

The function below, as its name suggests, checks the licence given the userID of the user:

```

• public function checkLicense($userId) {
•   $url = self::LICENSE_SERVER_HOST . '/chromewebstore/
v1/licenses/' .
•       self::APP_ID . '/' . urlencode($userId);
•
•   $request = OAuthRequest::from_consumer_and_token(
•       $this->consumer, $this->token, 'GET', $url, array());

```

The request has to be signed by the user before it is verified at the server.

```

• $request->sign_request($this->signatureMethod, $this-
>consumer,
•
• $this->token);
•
• return $this->send_signed_get($request);
• }
• }
•
• try {
• $openid = new LightOpenID();
• $userId = $openid->identity;
• if (!isset($_GET['openid_mode'])) {
• // This section performs the OpenID dance with the
normal redirect. Use it
• // if you want an alternative to the popup UI.

```

We can either use a popup UI, or simply even the redirection method. Here we see that the redirect method has been used. A popup UI could be a better option from the user's point of view.

```

• if (isset($_GET['login'])) {
• $openid->identity = 'https://www.google.com/accounts/
o8/id';
• $openid->required = array('namePerson/first',
'namePerson/last',
• 'contact/email');
• header('Location: ' . $openid->authUrl());
• }
• } else if ($_GET['openid_mode'] == 'cancel') {
• echo 'User has canceled authentication!';
• } else {
• $userId = $openid->validate() ? $openid->identity :
'';
• $_SESSION['userId'] = $userId;
• $attributes = $openid->getAttributes();
• $_SESSION['attributes'] = $attributes;
• }
• } catch (ErrorException $e) {
• echo $e->getMessage();
• exit;
• }

```



```

•
• if (isset($_REQUEST['popup']) && !isset($_SESSION['redirect_to'])) {
•     $_SESSION['redirect_to'] = $selfUrl;
• echo '<script type = "text/javascript">window.close();</script>';
• exit;
• } else if (isset($_SESSION['redirect_to'])) {
•     $redirect = $_SESSION['redirect_to'];
• unset($_SESSION['redirect_to']);
•
•
•
• header('Location: ' . $redirect);
• } else if (isset($_REQUEST['queryLicenseServer'])) {
•     $ls = new LicenseServerClient();
• echo $ls->checkLicense($_REQUEST['user_id']);
• exit;
• } else if (isset($_GET['logout'])) {
•     unset($_SESSION['attributes']);
•     unset($_SESSION['userId']);
•     header('Location: ' . $selfUrl);
• }
• ?>

```

Here's the HTML part:

```

• <!DOCTYPE html>
• <html>
• <head>
• <meta charset="utf-8" />
• <link href="main.css" type="text/css" rel="stylesheet" />
• <script type="text/javascript" src="popuplib.js"></script>
• <script type="text/html" id="ls_tmpl">
• <div id="access-level">
• <% if (result.toLowerCase() == 'yes') { %>
•     This user has <span class="<%= accessLevel.
toLowerCase() %>"><%= accessLevel %></span> access to this
application ( appId: <%= appId %> )
• <% } else { %>
•     This user has <span class="<%= result.toLower-
Case() %>"><%= result %></span> access to this application

```

```
( appId: <%= appId %> )
• <% } %>
• </div>
• </script>
• </head>
• <body>
• <nav>
```

If the userID is set, then the user is welcomed into the app; else, he/she finds the login box.

```
• <?php if (!isset($_SESSION['userId'])): ?>
• <a href="javascript:" onclick="openPopup(450, 500,
this);">Sign in</a>
• <?php else: ?>
• <span>Welcome <?php echo @$_SESSION['attributes']
['namePerson/first'] ?><?php echo @$_SESSION['attributes']
['namePerson/last'] ?> ( <?php echo $_SESSION['attributes']
['contact/email'] ?> )</span>
• <a href="?logout">Sign out</a>
• <?phpendif; ?>
• </nav>
• <?php if (isset($_SESSION['attributes'])): ?>
• <div id="container">
```

When the form is submitted, the licence server is queried to know what type of membership the user has. Accordingly, the access levels are set.

```
• <form action="<?php echo "$selfUrl?queryLicenseServer"
?>" onsubmit="return queryLicenseServer(this);">
• <input type="hidden" id="user _ id" name="user _ id"
value="<?php echo $_SESSION['userId'] ?>" />
• <input type="submit" value="Check user's access" />
• </form>
• <div id="license-server-response"></div>
• </div>
• <?phpendif; ?>
• <script>
• // Simple JavaScript Templating
• // John Resig - http://ejohn.org/ - MIT Licensed
• (function(){
• var cache = {};
```

```

    this.tmpl = function tmpl(str, data){
    // Figure out if we're getting a template, or if
    we need to
    // load the template - and be sure to cache the
    result.
    varfn = !/\W/.test(str) ?
    cache[str] = cache[str] ||
    tmpl(document.getElementById(str).innerHTML) :

    // Generate a reusable function that will
    serve as a template
    // generator (and which will be cached).
    new Function("obj",
        "var p=[],print=function(){p.push.
    apply(p,arguments);};" +

    // Introduce the data as local variables
    using with({})

        "with(obj){p.push('" +

    // Convert the template into pure JavaS-
    cript
    str

        .replace(/\r\t\n/g, " ")
        .split("<%").join("<t")
        .replace(/((^|>)[^t]*)'/g, "$1\r")
        .replace(/\t=(.*?)%/g, "', $1, '")
        .split("<t").join("'");")
        .split("%>").join("p.push('")
        .split("\r").join("\'")
        + "');}return p.join('');");

    // Provide some basic currying to the user
    return data ? fn( data ) : fn;
    };
    })();

    functionqueryLicenseServer(form) {
    varuserId = form.user_id.value;

    if (!userId) {

```

```

• alert('No OpenID specified!');
• return false;
• }
•
• var req = new XMLHttpRequest();
• req.onreadystatechange = function(e) {
•   if (this.readyState == 4) {
•     var resp = JSON.parse(this.responseText);
•     var el = document.getElementById('license-server-
response');
•     if (resp.error) {
•       el.innerHTML = ['<div class="error">Error ', resp.error.
code,
•         ': ', resp.error.message,
•       '</div>'].join('');
•     } else {
•       el.innerHTML = tmpl('ls_tmpl', resp);
•     }
•   }
• };
• var url =
•   [form.action, '&user_id=',
encodeURIComponent(userId)].join('');
• req.open('GET', url, true);
• req.send(null);
•
• return false;
• }
•
• function openPopup(w, h, link) {
•   var extensions = {
•     'openid.ns.ext1': 'http://openid.net/srv/ax/1.0',
•     'openid.ext1.mode': 'fetch_request',
•     'openid.ext1.type.email': 'http://axschema.org/
contact/email',
•     'openid.ext1.type.first': 'http://axschema.org/
namePerson/first',
•     'openid.ext1.type.last': 'http://axschema.org/
namePerson/last',
•     'openid.ext1.required': 'email,first,last',
•     'openid.ui.icon': 'true'

```

```

•     };
•
•     var googleOpener = popupManager.createPopupOpener({
•     opEndpoint: 'https://www.google.com/accounts/o8/ud',
•     returnUrl: '<?php echo "$selfUrl?popup=true" ?>',
•     onCloseHandler: function() {
•     window.location = '<?php echo $selfUrl ?>';
•     },
•     shouldEncodeUrls: false,
•     extensions: extensions
•     });
•     link.parentNode.appendChild(
•     document.createTextNode('Authenticating...'));
•     link.parentNode.removeChild(link);
•     googleOpener.popup(w, h);
•     }
• </script>
• </body>
• </html>

```

SCAN THE **QR CODE** OR GO TO
[BIT.LY/CHROMECHALLENGE](http://bit.ly/chromechallenge)
 AND STAND A CHANCE TO WIN
 SAMSUNG GALAXY TABS



SUMMARY

We will now look at what we have read so far, and advice you on how you should implement your app; and list it in the store. More importantly, we will be taking a look at the rating guidelines for your app as well as the developer agreement you enter with Google when you upload your app on the Chrome Web Store.

Best practices

Needless to say, in order to have a successful app, you need to design a great app. Interestingly, this isn't as hard as it sounds. The Chrome Web Store is still in its infancy. With ecosystems such as Android and several others growing at such a rapid rate, you'd get ideas to several new apps. Who knows, maybe your app just has what it takes to be the next Angry Birds of the web!

It is highly recommended to provide at least a modicum amount of support for Google accounts. Most of your users would already be logged into a Google account whenever they use your app. You'll also save them the misery of having to login multiple times to access your app; they, in turn, would thank you for a better user's experience. Another good idea is to correlate the Google account to the user account in your system by storing the user's OpenID credentials from Google's OpenID service.

Sometimes, users just get unsubscribed from apps unintentionally, for no fault of theirs. Hence, it only makes sense to retain user information for a few days after they cancel their subscription or uninstall your app. Even if users intentionally unsubscribe or uninstall your app, there is still some chance that they just might want to come back.

Also, do ponder over your app's listing on the Web Store. Be creative and descriptive when it comes to choosing your app's name, writing the description and designing the logo. Include plenty of screenshots and be very careful in choosing your app's category. The Chrome Developer Dashboard lets you specify one or two categories for each web app. Although the categories aren't final, do watch out for new categories that might be continuously added and could be a better fit for your web app. The following list describes each category that you can choose for your web app:

1. **Education:** Apps that impart knowledge to the user.
2. **Entertainment:** All leisure apps to keep users in the fun zone.
3. **Games:** All kinds of games that users can play.
4. **Lifestyle:** Apps for everyday life.
5. **News & weather:** Apps that discuss news, current affairs and weather.
6. **Productivity:** Apps that could enhance your professional life.
7. **Shopping:** Apps that could help you with deals, and shopping advice.
8. **Social & communication:** Apps that help people connect.
9. **Utilities:** Useful apps that have a narrower scope than the ones in the Productivity category.

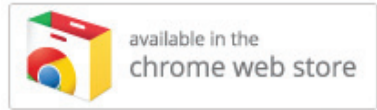
If you think, your app doesn't fit any of these categories; you can use the "Additional categories" field in the dashboard to suggest categories.

Branding guidelines

When you are naming and describing any of your applications, you should take care of the following guidelines. These guidelines will come into play whenever you use Google trademarks, and are subject to Google's approval and discretion.

Chrome Web Store Badge

The best way to popularise your app (or even extension or theme) on the Chrome Web Store is by using the "Available in the



Badge of honour

Chrome Web Store" badge on your site. Although you don't need Google's approval for this, you're expected to follow the following rules and regulations:

- Make sure the badge is legible, and prominently visible.
- Don't alter or modify the badge in any way. You're only permitted to resize, but not modify it in any way.
- Even while resizing the badge, maintain its aspect ratio.
- The badge shouldn't be the primary element on your page.
- Keep some distance between the badge and other logos and icons on your page.
- Don't use the badge on a page that contains or displays adult content, promotes gambling, promotes violence, contains hate speech, involves the sale of tobacco or alcohol to persons below the age of 21, violates other applicable laws or regulations or is otherwise objectionable.
- Make sure that clicking the badge always links to your page in the Chrome Web Store, and that your app, extension or theme is available in the store at all times that you use the badge.

Application / Developer Name and Logo

You can't use any Google trademarks or permit any resemblance with the name of your application, company or logo for your application, without written permission from Google.

Application Description

If your product is compatible with a Google product, you can make reference



What your app shouldn't look like

to that Google product by using the text “for”, “for use with”, or “compatible with”. Be sure to include the ™ symbol with the Google trademark.

Proper Attribution

You are expected to give proper attribution to Google for any use of their marks in your application title or in your description.

Rating Guidelines

Content ratings help users determine whether your application and its content are mature in nature. We recommend that you rate your application appropriately. When determining your app's rating, consider all the content in your app, including user generated content and ads, and content that your app links to.

The following rules will help you in determining whether your app should be rated as “Mature.”

Strong Language

Rate apps that contain explicit or frequent use of profanity as “Mature.”

Violence

Graphic violence or fantasy violence in an app should be treated as Mature content. Gratuitous real violence is not allowed in the Chrome Web Store.

ATD

Apps that focus on the consumption or sale of drugs, alcohol or tobacco should be rated “Mature.” Illegal activity or alcohol, tobacco or drug content that is targeted at minors is not allowed in the Chrome Web Store.

Sexual Content

Apps that focus on sexual content or sexually suggestive content should be rated “Mature.” Pornography and sexually explicit content are not allowed in the Chrome Web Store.

Examples:

Not mature

- Images of people kissing and hugging

- Dating app that is not focused on sexual relations
- App to help a woman choose the right bra size

Mature

- Close-up images of people wearing thongs
- Apps with sexual themes like adult “Truth or Dare” apps
- Sex position app with no porn, but has detailed descriptions


Developer program policies

Content Policies

The Google Chrome Developer content policies apply to your product’s content, including any ads it shows to users and any user-generated content it hosts or links to. After you’ve followed these guidelines and have rated your app accordingly, you should also look at the following:

1. Your product should not violate third party terms of service, or enable the unauthorised download of streaming content or media.
2. Don’t transmit viruses, worms, defects, Trojan horses, malware, or any other items of a destructive nature.
3. Your app must solely run in the browser and should be launchable either via Chrome’s launcher or by clicking on a standard URL.
4. No unauthorised publishing of people’s private and confidential information, such as credit card numbers, Social Security numbers, driver’s and other license numbers, or any other information that is not publicly accessible.
5. You should not infringe on the intellectual property rights of others, including patent, trademark, trade secret, copyright, and other proprietary rights.

Spam

As a responsible developer, you’re expected to stay free from spam. When we say spam, we mean refraining from posting repetitive content or misleading app descriptions that attempt to manipulate ranking or relevancy in the store’s search results. Tricks such as rating an application multiple times in order to change the placement of any product, are actively discouraged. 

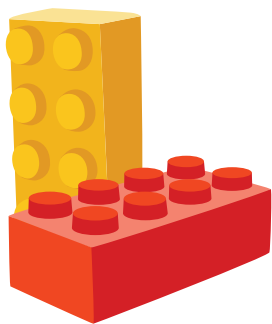
SCAN THE **QR CODE** OR GO TO
[BIT.LY/CHROMECHALLENGE](https://bit.ly/chromechallenge)
 AND STAND A CHANCE TO WIN
 SAMSUNG GALAXY TABS



[illegible]

[illegible]

[illegible]



Chrome is simple.



chrome by Google

Download now at www.google.co.in/chrome

digit brings you



Chrome Challenge

5 JUST FOLLOW SIMPLE STEPS AND

GET FEATURED IN THE DIGIT MAGAZINE

- 1 **REGISTER FOR THE CONTEST ON**
bit.ly/chromechallenge
- 2 Download tools and documents you need from the Digit DVD that you got along with this issue
- 3 Create your Chrome App, preferably in the lifestyle or the entertainment space
- 4 Publish your app on the Chrome Webstore at *bit.ly/chromeupload*
- 5 Notify us by sending a mail to chrome@thinkdigit.com with details of Chrome app that you submitted as part of this contest



Best App

Gets featured in the September Issue of Digit Magazine

Top-5 apps

Gets one Samsung Galaxy Tab each



Assured Prize

Every participant who submits a valid app would be given a six-months subscription to Digit worth ₹1000





Chrome is fast.



chrome by Google

Download now at www.google.co.in/chrome